

## СОДЕРЖАНИЕ

Приложение А Архитектура 32-разрядных микропроцессоров семейства x86 .....	2
А.1 Состав устройств и режимы работы .....	2
А.2 Программная модель .....	3
А.3 Назначение бит регистра флагов .....	7
А.4 Типы данных .....	8
А.5 Режимы (методы) адресации .....	10
А.6 Команды процессоров 386+ .....	13
А.6.1 Команды передачи данных .....	13
А.6.2 Арифметические команды .....	15
А.6.3 Логические команды .....	19
А.6.4 Команды сдвигов .....	19
А.6.5 Команды битовых операций .....	20
А.6.6 Команды обработки цепочек .....	21
А.6.7 Команды работы с флагами (флажковые команды) .....	23
А.6.8 Команды передачи управления .....	24
Приложение Б Архитектурное расширение 32-разрядных микропроцессоров – процессор чисел с плавающей точкой (FPU) .....	28
Б.1 Форматы чисел с плавающей точкой .....	28
Б.2 Регистры сопроцессора x87 .....	32
Б.3 Команды сопроцессора x87 .....	36
Б.3.1 Команды передачи данных сопроцессора x87 .....	37
Б.3.3 Дополнительные арифметические команды .....	40
Б.3.4 Команды сравнения .....	40
Б.3.5 Трансцендентные команды .....	41
Б.3.6 Команды управления сопроцессором x87 .....	45
Приложение В Значения сигнатуры идентификации CPU x86-64 .....	49
Приложение Г Флаги свойств процессоров x86-64 .....	51
Приложение Д Порядок разработки ассемблерной DOS-программы .....	55

## ПРИЛОЖЕНИЕ А

### АРХИТЕКТУРА 32-РАЗРЯДНЫХ МИКРОПРОЦЕССОРОВ СЕМЕЙСТВА X86

#### А.1 Состав устройств и режимы работы

Базовая архитектура 32-разрядных процессоров (обозначаемых 386+) является общей для существующих на данный момент процессоров фирмы INTEL – 386, 486, PENTIUM и т.д.

МП состоит из 3 основных частей: устройства обработки; устройства управления ЗУ; интерфейсного блока.

Устройство обработки состоит из исполнительного устройства (операционной части) и блока команд (управляющей части). Содержит 8 32-разрядных РОН, 64-битовый циклический сдвигатель. Умножение и деление осуществляется на 1 бит за цикл. Алгоритм умножения такой, что процесс прекращается, когда наиболее значащий бит, умножается на все нули. Типичное время умножения 32-разрядных чисел около 1 мкс (для процессора I80386).

Устройство управления ЗУ состоит из сегментного и страничного блоков. Сегментный блок позволяет работать с логическими адресами со всеми вытекающими отсюда преимуществами. Страничная организация используется внутри сегмента и управляет физическими адресами. Каждая задача может иметь до 16381 ( $2^{14}$ ) сегмента до 4 ГиБ каждый ( $2^{32}$ ), т.е. виртуальная память может быть размером 64 ТиБ ( $2^{46}$ ).

Интерфейсный блок обеспечивает взаимодействие с внешними устройствами, включая автоматическое управление разрядностью шины, и формирование сигналов активности байтов.

МП 386+ могут функционировать в трех режимах.

REAL ADDRESS MODE – режим реальной адресации (PPA) – характеризуется тем, что МП работает как очень быстрый 8086 с 32-битовым расширением; в этом режиме возможна адресация 1 МиБ физической памяти (на самом деле, как у I80286, - почти на 64 КиБ больше).

PROTECTED ADDRESS MODE – режим защищенной виртуальной адресации (PBA) – реализует все достоинства МП (режим параллельного выполнения нескольких задач несколькими 8086 – по одному на задачу). На одном процессоре в таком режиме могут одновременно исполняться несколько задач с изолированными друг от друга реальными ресурсами. При этом использование физического адресного пространства памяти управляется механизмами сегментации и трансляции страниц. Попытки выполнения недопустимых команд, выхода за рамки отведенного пространства памяти и разрешенной области ввода-вывода контролируются системой защиты.

VIRTUAL 8086 MODE – режим виртуального процессора 8086 (сокращенно – V86). Прикладная программа, которая выполняется в этом режиме, полагает, что она работает на процессоре 8086. Однако, некоторые

команды, в основном связанные с управлением вводом-выводом, программе выполнять запрещается, поэтому при нарушении защиты генерируется прерывание и управление передается операционной системе.

Основная структура в организации памяти – сегмент. Сегменты – блоки памяти переменной длины (от 1 Б до 4 Гиб), имеющие определенные атрибуты. Три основных типа сегментов – СТЕК, КОМАНДЫ, ДАННЫЕ.

Система команд включает 9 групп команд:

- передачи данных;
- арифметические и логические;
- сдвига;
- обработки строк;
- манипуляции битами;
- передачи управления;
- поддержки языков высокого уровня;
- поддержки операционной системы;
- управления процессором.

Команды могут содержать от 0 до 3 операндов, размещенных в регистрах, памяти или непосредственно в команде. Большинство безоперандных команд – однобайтовые. Однооперандные команды обычно – двухбайтовые. Средняя длина команды – 3,2 Б. Это позволяет хранить в среднем 5 команд в 16-байтовой ОЧЕРЕДИ КОМАНД БЛОКА ОПЕРЕЖАЮЩЕЙ ВЫБОРКИ.

При использовании двух операндов возможны следующие типы взаимодействия:

- регистр-регистр;
- память-регистр;
- регистр-память;
- непосредственный операнд-регистр;
- непосредственный операнд-память;
- память-память.

Операнды могут быть 8, 16 или 32-разрядными. Когда выполняются команды, написанные для 386+, операнды имеют длину 8 или 32 бита, когда для 80286 и 8086 – 8 или 16 бит. Ко всем инструкциям могут добавляться префиксы, которые изменяют длину операндов (т.е. позволяют использовать 32-битовые операнды в 16-битовых командах или 16-битовые операнды в 32-битовых командах).

## А.2 Программная модель

МП 386+ имеют несколько десятков регистров, разбитых на следующие группы:

- регистры общего назначения;
- сегментные регистры;
- указатель команд и регистр флагов (признаков);
- управляющие регистры;

- регистры системных адресов;
- отладочные регистры;
- тестовые регистры.

Набор регистров общего назначения (рис. А.1) включает соответствующие регистры процессоров I8086 и I80286. Все эти регистры, кроме сегментных, имеют разрядность 32 бита и к прежнему обозначению их имен добавилась приставка «Е» (Extended – расширенный). Отсутствие приставки «Е» в имени означает ссылку на младшие 16 бит расширенных регистров. Обратиться к старшим 16 битам расширенных регистров ни одна команда не может. Как и в I8086, возможно независимое обращение к младшему и старшему байтам регистров AX, BX, CX, DX.

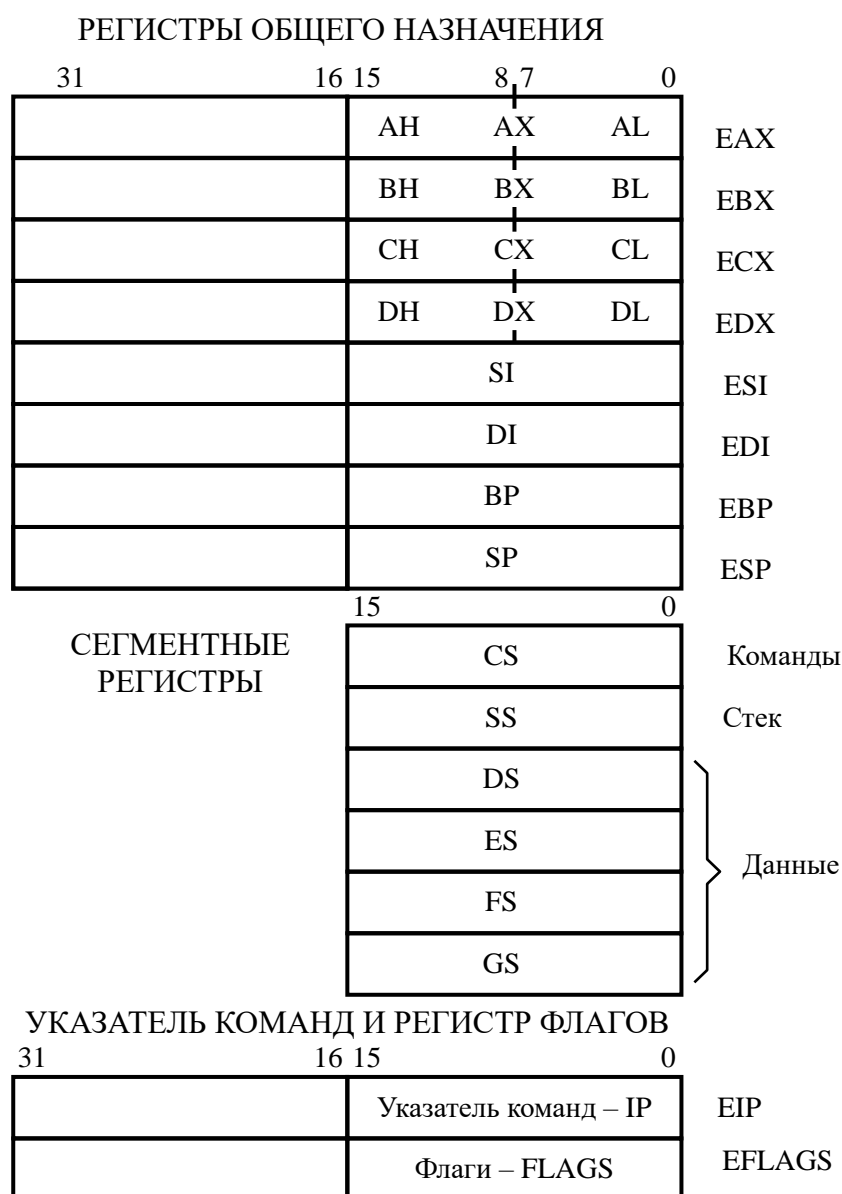


Рис. А.1 – Регистры 32-разрядных МП 386+

Архитектура МП 386+ позволяет непосредственно обращаться к 6 сегментам (размером до 4 Гиб каждый) при помощи специальных селекторов, которые загружаются в сегментные регистры программно. Содержимое РОНов, селекторов, указателя команд и регистра флагов

(признаков) зависит от выполняемой задачи и автоматически перегружается при переключении задач.

Остальные регистры МП используются, главным образом, для упрощения проектирования и отладки операционной системы.

Регистры общего назначения (РОН) – используются для хранения операндов и адресов. Могут работать с операндами, имеющими длину 1, 8, 16, 32 и 64 бита или с битовыми полями длиной от 1 до 32 бит.

Указатель команд – EIP – хранит смещение, которое всегда складывается со значением кодового сегментного регистра (CS) и определяет адрес следующей команды. При 16-битовой адресации используются только младшие 16 бит (IP).

МП 386+ содержат 6 16-битовых сегментных регистров (у предыдущих поколений – только 4 сегментных регистра), хранящих значение селектора и определяющих значения начальных (базовых) адресов сегментов. В РВА каждый сегмент может изменяться в диапазоне от одного байта до максимального значения физического адресного пространства 4 ГиБ. В РРА размеры сегмента ограничены размером 64 КиБ.

Для компактного кодирования команд и повышения производительности МП – команды не содержат явного указания на используемый сегментный регистр. Определение сегментного регистра (по умолчанию) производится автоматически в соответствии с табл. А.1. Сегментные регистры FS и GS – не выбираются по умолчанию ни в одной команде и могут быть выбраны только префиксом замены сегмента.

Обычно название сегментного регистра указывает на тип информации, для адресации которой он используется. Применение префикса переадресации позволяет явно определять используемый сегментный регистр (см. название регистров в скобках во второй колонке табл. А.12), в том числе FS и GS.

Таблица А.1 – Выбор сегментных регистров и внутрисегментного смещения

Тип обращения к памяти	Сегментный регистр	Смещение
Выборка команды	CS	EIP (IP)
Обращение к стеку	SS	ESP (SP)
Адресация операнда	DS (CS,SS,ES,FS,GS)	EA
Элемент цепочки-источника	DS (CS,SS,ES,FS,GS)	ESI (SI)
Элемент цепочки-приемника	ES	EDI (DI)
Операнд с использованием в качестве базового регистра		
EBP (BP) или ESP (SP)	DS (CS,SS,ES,FS,GS)	EA

Дескрипторные регистры сегментов программно не видимы, но они неразрывно связаны с соответствующими сегментными регистрами (рис. А.2). Каждый дескрипторный регистр хранит 32-битовый базовый адрес сегмента, 20-битовый размер сегмента и другие необходимые атрибуты сегмента. Когда значение селектора загружается в сегментный регистр, в

режиме виртуальной адресации (РВА) соответствующий дескрипторный регистр автоматически загружается информацией из дескрипторной таблицы.

В РВА базовый адрес, размер и атрибуты сегментного дескриптора определяется селектором. 32-битовый **БАЗОВЫЙ АДРЕС** сегмента

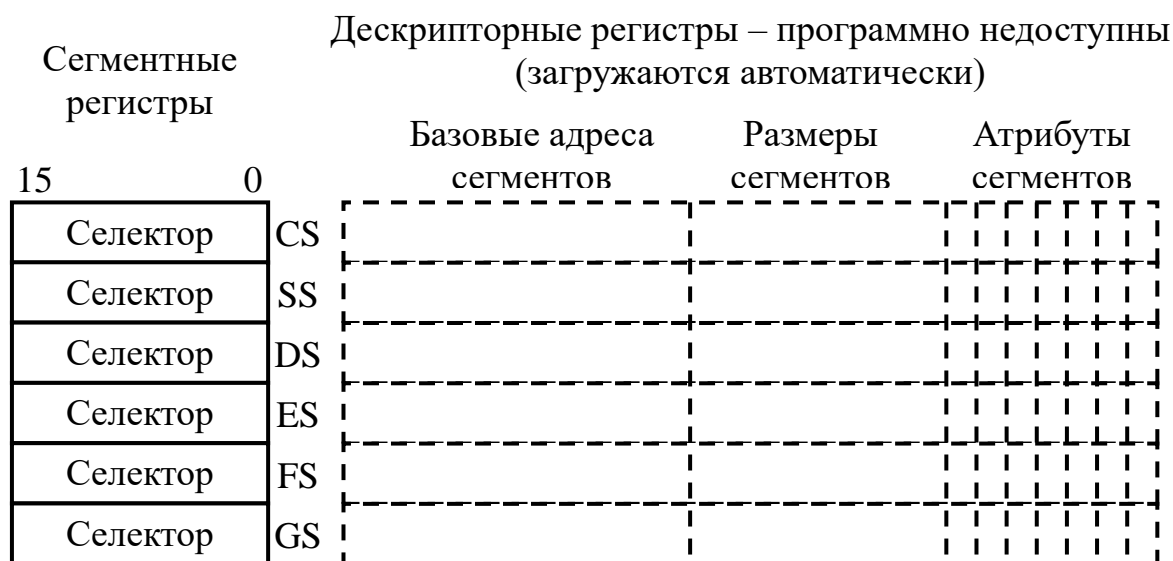


Рис. А.2 – Сегментные регистры и соответствующие дескрипторные регистры МП 386+

становится компонентом формирования исполнительного адреса, 20-битовый **РАЗМЕР СЕГМЕНТА** используется для проверки границ рабочей области, а **АТРИБУТЫ** проверяются на соответствие типу запрашиваемой памяти (типу обращения).

В РРА непосредственно используется только базовый адрес (со сдвигом на 4 разряда влево), а размеры сегмента и атрибуты постоянны (фиксированы для РРА).

**РЕГИСТР ФЛАГОВ** (признаков) – EFLAGS (рис. А.3) – отражает состояние МП. При использовании только 16 младших разрядов - регистр флагов совместим с предыдущими моделями МП. На рис. А.3 обозначены символами: х – системный флаг; s – флаг состояния; с – управляющий флаг.

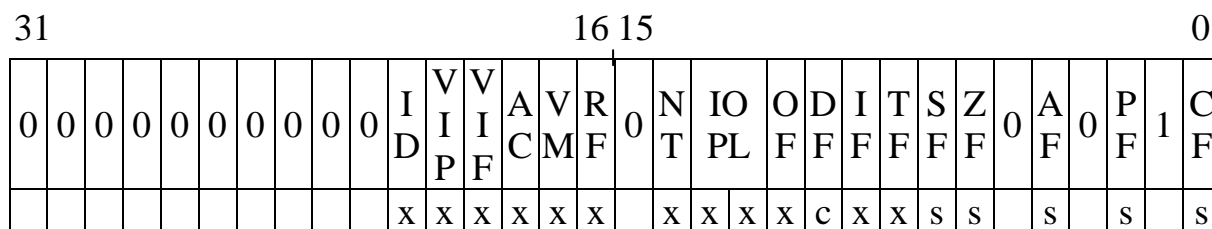


Рис. А.3 – Регистр флагов EFLAGS

### А.3 Назначение бит регистра флагов

CF - (Carry Flag) – флаг переноса, показывающий перенос (заем) из старшего бита при арифметических операциях, а также значение выдвигаемого бита при сдвиге операнда;

AF - (Auxiliary Flag) – флаг вспомогательного переноса (заема) в младшей тетраде для десятичной арифметики;

OF - (Overflow Flag) – флаг арифметического переполнения, определяющий (при OF=1) выход знакового результата за границы диапазона;

ZF - (Zero Flag) – флаг нуля, показывающий (при ZF=1) нулевой результат команды;

SF - (Sign Flag) – флаг знака, дублирует значение старшего бита результата, который при использовании дополнительного кода соответствует знаку числа;

PF - (Parity Flag) – флаг паритета (четности), фиксирующий (при PF=1) наличие в младшем байте результата четного числа единичных бит.

IOPL (Input/Output Privilege Level) - используется только в РВА. IOPL указывает максимальную величину текущего приоритета, обеспечивающую выполнение команд ввода-вывода без реакции на 13 ошибку. Этот признак также обеспечивает выбор IF, когда новое значение выталкивается из стека в регистр признаков. POPF и IRET могут изменять IOPL поле, когда IOPL = 0 (CPL=0). При переключении задач IOPL может изменяться всегда при переписи TSS (286+).

NT - (Nested Task Flag) – флаг вложенной задачи (286+);

ID (Id Flag) – флаг доступности команды идентификации CPUID (PENTIUM+ и некоторые 486+);

VIP (Virtual Interrupt Pending) – виртуальный запрос прерывания (PENTIUM+);

VIF (Virtual Interrupt Flag) – виртуальная версия флага IF (разрешения прерывания) для многозадачных систем (PENTIUM+).

AC (Alignment Check) – флаг контроля выравнивания. При исполнении программ на уровне привилегий 3 в случае обращения к операнду, не выровненному по соответствующей границе (2, 4, 8 байт), и при установленном флаге AC произойдет исключение-отказ 17 с нулевым кодом ошибки. На уровнях привилегий 0, 1, 2 контроль выравнивания не производится (486+).

VM (Virtual 8086 Mode) – обеспечивает режим виртуального 8086 внутри режима виртуальной адресации. При VM = 1 МП будет переключен в режим виртуального I8086, при этом управление перезагрузкой сегментов будет осуществляться подобно I8086, но с исключением 13 недействительных привилегированных команд. VM может быть установлен в РВА командой IRET (если уровень приоритета = 0) и задача переключается на более низший уровень. Команда POPF не влияет на VM. Команда PUSHF всегда сбрасывает VM в 0, если она выполняется в режиме виртуального

8086. Содержимое регистра признаков будет копироваться при прерываниях или сохраняться при переключении задачи, если прерывание будет выполняться в режиме виртуального 8086 (386+).

RF (Resume Flag) – флаг возобновления – используется совместно с отладочными регистрами контрольных точек (прерываний) или пошагового режима. С его помощью проверяется ход выполнения команд в отладочном режиме (процесс отладки). Если установлен  $RF = 1$ , то это позволяет игнорировать ошибки, возникающие при отладке до следующей команды. RF автоматически сбрасывается в 0 при успешном выполнении команды (ошибки не обнаружены), за исключением команд IRET и POPF, а также JMP, CALL и INT при переключении задач. Эти команды устанавливают RF в состояние, определяемое состоянием памяти. Например, в конце выполнения подпрограммы обслуживания контрольной точки команда IRET может установить RF в состояние, соответствующее значению регистра признаков, хранимого в стеке без повторной установки RF в 1 (386+).

NT (Nested Task Flag) – флаг вложенной задачи (гнездования) используется только в режиме виртуального адреса (PBA).  $NT=1$  указывает, что текущая задача является вложенной по отношению к другой задаче. Этот бит устанавливается и сбрасывается при вызове других задач. NT проверяется командой IRET для определения внутри заданного или внешнего по отношению к данной задаче возврата. Команды POPF и IRET будут устанавливать NT в соответствии с тем, что хранится в стеке для любого уровня привилегированности (286+).

#### А.4 Типы данных

32-разрядные процессоры фирмы INTEL (386+) работают с целыми двоичными числами длиной 8, 16 или 32 бита и двоично-кодированными десятичными числами (BCD-числами) длиной 8 бит. Двоичные числа допускают интерпретацию как целых без знака и целых со знаком, а десятичные (BCD) – знака не имеют.

В ДВОИЧНЫХ ЦЕЛЫХ ЧИСЛАХ БЕЗ ЗНАКА все разряды считаются значащими (рис. А.4). ДВОИЧНЫЕ ЦЕЛЫЕ ЧИСЛА СО ЗНАКОМ представляются в дополнительном коде. Старший бит является знаковым:  $S = 0$  – число положительное,  $S = 1$  – число отрицательное.

ДЕСЯТИЧНЫЕ ЧИСЛА представляются в упакованном и неупакованном форматах. Упакованный формат предполагает, что байт содержит две десятичные цифры в коде с весами 8421, занимающих младшую и старшую тетрады. Диапазон представимых BCD-чисел – 0...99. В неупакованном формате байт содержит одну десятичную цифру, которая обычно изображается в символьном коде ASCII.

Новые команды процессоров 386+ поддерживают БИТОВЫЕ ДАННЫЕ:

- БИТ – одиночный двоичный разряд.
- БИТОВОЕ ПОЛЕ – группа до 32 битов.



– ЦЕПОЧКА БИТОВ (СТРОКА) – набор последовательных битов, длиной до 4 Гбит.

Процессор может легко оперировать с цепочками бит, байт, слов и двойных слов. Под ЦЕПОЧКОЙ (string) понимается последовательность практически любой длины отдельных, но взаимосвязанных элементов данных, ХРАНЯЩИХСЯ ПО СОСЕДНИМ АДРЕСАМ.

УКАЗАТЕЛИ применяются для обращения к некоторым объектам в памяти, например, адресам подпрограмм. Близкие (NEAR) или внутрисегментный указатель (рис. А.4) – это 16-битовое или 32-битовое смещение внутри текущего сегмента. Далекий (FAR) или межсегментный указатель применяется в тех случаях, когда программа осуществляет передачу управления в другой сегмент. Такой указатель определяет новый сегмент (с помощью селектора) и 16-или 32-битовое смещение внутри этого сегмента.

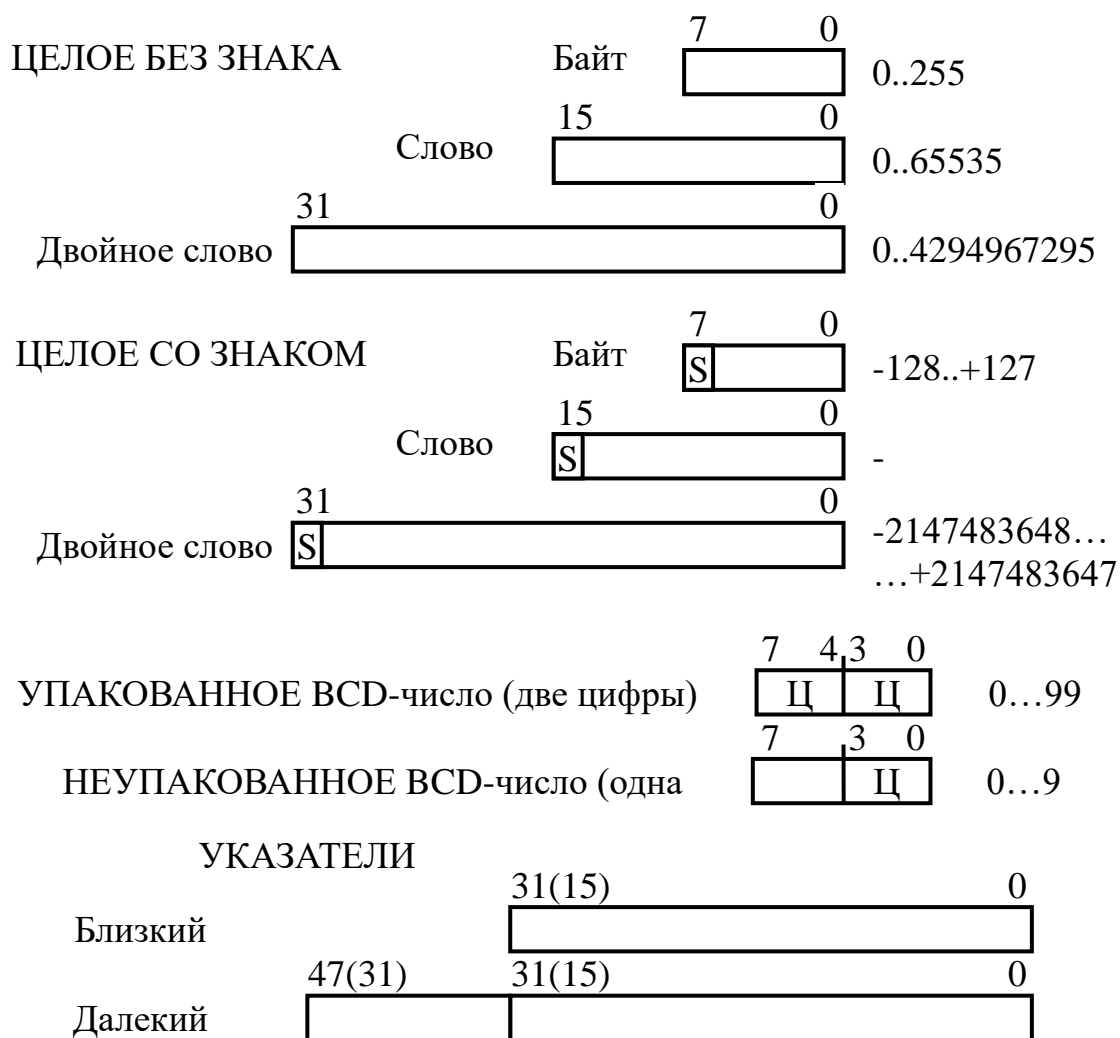


Рис. А.4 – Типы данных 32-разрядных процессоров (386+)

При размещении операндов в памяти необходимо учитывать, что процессоры 386+ не накладывают ограничения на размещение данных. Однако производительность процессора повышается, если слова размещены по четным адресам, а двойные слова – по адресам, кратным четырем. Такой

принцип называется **ВЫРАВНИВАНИЕ АДРЕСОВ** по границам слов или двойных слов. Выравнивание особенно важно для стека, который работает только со словами или двойными словами.

#### А.5 Режимы (методы) адресации

Процессоры 386+ обеспечивает 13 режимов адресации, которые рассчитаны на эффективное выполнение программ, написанных на языках высокого уровня (ЯВУ) типа: C++, Фортран и др..

**НЕЯВНАЯ АДРЕСАЦИЯ.** Операнд адресуется неявно, если в команде нет специальных полей для его определения, т.е. операнд задается полем команды. В ассемблерных кодах с неявной адресацией поле операнда пустое. Примеры команд с неявной адресацией:

```
AAA      ; Коррекция р-а AL после сложения
CMC      ; Инверсия флага переноса
STD      ; Установить в 1 флаг направления.
```

**РЕЖИМ РЕГИСТРОВОЙ АДРЕСАЦИИ** и **РЕЖИМ НЕПОСРЕДСТВЕННОЙ АДРЕСАЦИИ** – предназначены, соответственно, для адресации одного из регистров регистрового блока или непосредственного операнда в команде с разрядностью 8, 16 или 32 бита:

```
INC      esi      ; Инкремент р-ра ESI
SUB      ECX, ECX ; Сбросить р-р ECX
MOV      EAX, CR0 ; Передать в EAX содержимое CR0.
MOV      EAX, 0F0F0F0Fh ; Загрузить константу в EAX
AND      AL, 0FH  ; Выделить младшую тетраду р-ра AL
BT       EDI, 3   ; Передать во флаг CF третий бит р-ра EDI
```

Имеется 10 режимов **АДРЕСАЦИИ ПАМЯТИ**. Исполнительный адрес включает в себя два компонента адреса ячейки памяти – сегмент и эффективный адрес (внутрисегментное смещение). **ЭФФЕКТИВНЫЙ АДРЕС (ЕА)** вычисляется суммированием следующих элементов:

– **СМЕЩЕНИЕ** (смещение) – целая 8-ми или 32-битовая величина со знаком, непосредственно задаваемая в команде (16-битовые смещения могут использоваться при помощи префикса);

– **БАЗА** – содержимое любых РОНов. Базовые регистры обычно используются компиляторами в качестве точки отсчета локальной области памяти;

– **ИНДЕКС** – содержимое любых РОНов, исключая ESP. Индексные регистры используются для доступа к элементам строк или массивов;

– **МНОЖИТЕЛЬ f** - указывает шаг (1, 2, 4 или 8) для индексного регистра. Шаг индексации позволяет успешно адресовать массивы или структуры, содержащие многобайтовые операнды.

$$EA = \text{БАЗА} + \text{ИНДЕКС} * (\text{ШАГ ИНДЕКСАЦИИ}) + \text{СМЕЩЕНИЕ}.$$

Вычисление эффективного адреса (ЕА) практически не ухудшает производительность процессора из-за использования конвейерного режима.

**ПРЯМАЯ АДРЕСАЦИЯ** – смещение адреса операнда содержится в 8, 16 или 32 разрядах команды:

```
MOV    AL, [2000h]          ; Передать байт в регистр AL
INC    dword prt [123456h]  ; Инкремент двойного слова
                                ; в памяти.
```

**РЕГИСТРОВЫЙ КОСВЕННЫЙ МЕТОД АДРЕСАЦИИ** – базовый или индексный регистр содержат адрес операнда :

```
MOV    AL, [ECX]            ; Передать в р-р AL байт по
                                ; адресу из ECX
DEC    word prt [ESI]       ; Декремент слова по адресу из ESI.
```

**БАЗОВАЯ АДРЕСАЦИЯ** – базовый регистр суммируется со смещением:

```
MOV    EAX, [EBX+4]         ; Передать двойное слово из памяти
ADD    [ECX+10h], DX        ; Прибавить к слову в памяти.
```

**ИНДЕКСНАЯ АДРЕСАЦИЯ** – индексный регистр (любой РОН кроме ESP) суммируется со смещением:

```
SUB    array[ESI], 2        ; Вычесть 2 из элемента массива
IMUL    vector[ECX]         ; Умножить EAX на элемент массива.
```

**ИНДЕКСНАЯ АДРЕСАЦИЯ С ШАГОМ** – содержимое индексного регистра умножается на шаг «f» и суммируется со смещением:

```
MOV    EAX, vec[ECX*4]      ; Переслать в EAX двойное слово
                                ; из массива.
```

**БАЗОВО-ИНДЕКСНАЯ АДРЕСАЦИЯ.** –  $EA = \text{БАЗА} + \text{ИНДЕКС}$ :

```
ADD    EAX, [EBX][ESI]      ; Прибавить к EAX двойное
                                ; слово из памяти.
```

**БАЗОВО-ИНДЕКСНАЯ АДРЕСАЦИЯ С ШАГОМ.** –  $EA = \text{БАЗА} + \text{ИНДЕКС} * \text{ШАГ}$ :

```
INC    word prt [EDX][EDI*4] ; Инкремент ячейки памяти.
```

**БАЗОВО-ИНДЕКСНАЯ АДРЕСАЦИЯ СО СМЕЩЕНИЕМ.** –  
 $EA = \text{БАЗА} + \text{ИНДЕКС} + \text{СМЕЩЕНИЕ}$ :

MOV AX, [ECX][ESI+20h] ; Переслать слово из памяти

**БАЗОВО-ИНДЕКСНАЯ АДРЕСАЦИЯ СО СМЕЩЕНИЕМ И С ШАГОМ.** –  $EA = \text{БАЗА} + \text{ИНДЕКС} * \text{ШАГ} + \text{СМЕЩЕНИЕ}$ :

ADD AX, [EDX][EDI\*4+10h] ; Сложить AX с ячейкой памяти.

**СТЕКОВАЯ АДРЕСАЦИЯ** (можно рассматривать как вариант регистровой косвенной адресации) – в указателе стека ESP (SP) формируется 32-битовое (16-битовое) внутрисегментное смещение для операнда в стековом сегменте:

```
PUSH ECX      ; Включить в стек содержимое регистра
PUSHFD        ; Включить в стек содержимое EFLAGS
PUSH 4000h    ; Включить в стек константу
POP EDX       ; Извлечь из стека в регистр
POPFD         ; Извлечь из стека в регистр EFLAGS
POP [ESI]     ; Извлечь из стека в ячейку памяти
```

В таблице А.2 показана разница в использовании базовых и индексных регистров для 16- и 32-битовых адресов.

Для обеспечения совместимости ПО процессоров необходимо программы (с 16-битовыми командами МП 86 и 286) выполнять на МП 386+ в реальном или защищенном режимах. Процессор определяет размерность адреса, анализируя бит **D** (Default) в дескрипторе сегмента. Если D=0, то все длины операндов и эффективных адресов составляют 16 бит. Если D=1, – 32 бита. В реальном режиме – 16 бит.

Изменение размерности адреса и данных, задаваемых битом **D**, обеспечивают два префикса, выбираемые перед командами:

**ПРЕФИКС РАЗМЕРНОСТИ ОПЕРАНДА (OperandSize),**

**ПРЕФИКС ДЛИНЫ АДРЕСА (AddressSize).**

Наличие префикса коммутирует (переключает) размер операнда или размер эффективного адреса на значение, противоположное принимаемому по умолчанию (по биту **D**).

Префиксы могут использоваться совместно с любой инструкцией и в любом режиме – реальном, виртуальном и V86. Префикс длины адреса не обеспечивает размерность адреса более 64 КиБ в режиме реальной адресации. Адрес свыше 0FFFFh будет рассматриваться как ошибка.

Таблица А.2 – Базовые и индексные регистры для 16- и 32-битовых адресов

	16-битовый адрес	32-битовый адрес
Базовый регистр	BX, BP	Любой 32-битовый РОН
Индексный регистр	SI, DI	Любой 32-битовый РОН, исключая ESP
Шаг индексации «f»	нет	1, 2, 4, 8
Смещение	0, 8, 16 бит	0, 8, 32 бит

## А.6 Команды процессоров 386+

### А.6.1 Команды передачи данных

Предназначены для пересылок байт (обозначается В), слов (W) или двойных слов (D) из памяти в регистр, из регистра в память и из регистра в регистр. В одной команде невозможно использование двух операндов, расположенных в памяти (за исключением цепочечных команд и операций со стеком).

Команда **MOV** передает байт, слово или двойное слово из источника в приемник. В поле операндов приемник находится на первом месте, источник – на втором.

Команда **XCHG** осуществляет обмен байт, слов или двойных слов. Различий между приемником и источником нет.

Команда **XLAT** заменяет значение в регистре AL на байт из таблицы, адресуемой регистром (E)BX, причем индексом таблицы служит содержимое регистра AL. Эта команда удобна для преобразования из одного кода в другой.

Команда **LEA** обеспечивает вычисление эффективного адреса EA ячейки памяти в соответствии с указанным способом адресации и загрузку EA (а не содержимого адресуемой ячейки памяти!) в указанный общий регистр.

Команды **LDS, LES...** загружают четыре (или шесть) смежных байта из памяти в адресуемый регистр (16 или 32 бита) и в соответствующий сегментный регистр (16 бит). Слово (двойное слово) операнда источника из ячейки памяти, адресуемой в соответствии с указанным методом адресации, передается в выбранный регистр, а следующее слово – в регистр DS (команда LDS), в регистр ES (команда LES) и т.д.

В таблицах приняты следующие обозначения:

- src – операнд-источник;
- dest – операнд-назначение (операнд-приемник);
- reg – 8/16/32-битовый регистр;
- reg16/32 – 16/32-битовый регистр;
- reg16 – только 16-битовый регистр;
- reg32 – только 32-битовый регистр;
- mem – 8/16/32-битовая ячейка памяти, адресуемая регистрами процессо
- r/m – 8/16/32-битовый регистр или ячейка памяти, адресуемая регистрами процессора;
- r/m/i – 8/16/32-битовый регистр, ячейка памяти, адресуемая регистрами процессора или непосредственный операнд;
- addr – 16/32-битовый адрес;
- immед – непосредственный операнд.

Таблица А.3 – Команды пересылки данных

MOV dest, src	Пересылка (копирование) данных из регистра, памяти или непосредственного операнда в регистр или память
XCHG r/m, reg	Обмен данными (взаимный) между регистрами или регистром и памятью
BSWAP reg32	Перестановка байтов в регистре из порядка младший-старший в порядок старший-младший (486+)
MOVSXB reg, r/m	Копирование байта с расширением до слова или двойного слова, заполняя старшие биты знаком (386+)
MOVSXW reg, r/m	Копирование слова с расширением до двойного слова, заполняя старшие биты знаком (386+)
MOVZXB reg, r/m	Копирование байта с расширением до слова или двойного слова, заполняя старшие биты нулем (386+)
MOVZXW reg, r/m	Копирование слова с расширением до двойного слова, заполняя старшие биты нулем (386+)
XLAT	Трансляция (перекодирование) содержимого AL в значение из таблицы трансляции, адресуемой в (E)BX: $AL \leftarrow [(E)BX + AL]$
LEA reg16/32, mem	Загрузка эффективного адреса в регистр
LDS reg16/32, mem	Загрузка в регистр (двойного) слова из памяти, а в DS – следующего 16-битового слова
LES reg16/32, mem	Загрузка в регистр (двойного) слова из памяти, а в ES – следующего 16-битового слова
LFS reg16/32, mem	Загрузка в регистр (двойного) слова из памяти, а в FS – следующего 16-битового слова
LGS reg16/32, mem	Загрузка в регистр (двойного) слова из памяти, а в GS – следующего 16-битового слова
LSS reg16/32, mem	Загрузка в регистр (двойного) слова из памяти, а в SS – следующего 16-битового слова
IN AL(AX), port8	Ввод в AL (или AX,EAX) из порта с адресом port8
IN AL(AX), DX	Ввод в AL (или AX,EAX) из порта с адресом, хранящимся в DX
OUT port8, AL(AX)	Вывод из AL (или AX,EAX) в порт с адресом port8
OUT DX, AL(AX)	Вывод из AL (или AX,EAX) в порт с адресом, хранящимся в DX

Команда **PUSH** передает слово (или двойное слово) из источника в стек, а команда **POP** осуществляет противоположное действие – передает (двойное) слово из стека в приемник. Стек – это область памяти, в которой размещается текущий сегмент стека. Регистр (E)SP содержит смещение последнего включенного в стек слова; оно (смещение) называется **ВЕРШИНОЙ СТЕКА**. По мере включения в стек новых слов они располагаются по меньшим адресам памяти; говорят, что стек растет в направлении уменьшения адресов.

Таблица А.4 – Команды работы со стеком

PUSH r/m	Помещение (двойного) слова из регистра или памяти в стек
PUSH imm8	Помещение непосредственного операнда в стек (286+)
PUSHA (D)	Помещение в стек регистров AX,CX,DX,BX,SP,BP,SI,DI (286+) или их 32-битовых расширений (386+)
POP r/m	Извлечение (двойного) слова данных из стека в регистр или память
POPA (D)	Извлечение данных из стека в регистры DI,SI, BP,SP,BX,DX,CX,AX (286+) или их 32-битовых расширений (386+)
PUSHF (D)	Помещение в стек регистра флагов FLAGS (EFLAGS)
POPF (D)	Извлечение данных из стека в регистр флагов FLAGS (EFLAGS)

Команда **PUSH** начинается с уменьшения (декремента) содержимого регистра (E)SP на 2 (или 4), т.е. адресует следующее свободное слово (или двойное слово) в стеке; после чего передается (двойное) слово из источника.

Команда **POP** передает слово (или двойное слово) из стека в приемник и завершается увеличением (инкрементом) содержимого (E)SP на 2 (или на 4).

Команда **PUSHA (D)** включает в стек регистры в таком порядке: (E)AX, (E)CX, (E)DX, (E)BX, (E)SP, (E)BP, (E)SI, (E)DI. Включаемым значением регистра (E)SP является то его значение, которое было в нем до выполнения команды PUSHA (D). При выполнении команды PUSHA (D) происходит декремент содержимого регистра (E)SP на 2 (или на 4) при включении в стек каждого регистра.

Извлечение из стека, реализуемое командой **POPA (D)**, вызовет инкремент содержимого регистра (E)SP на ту же величину, поэтому команде POPA (D) не требуется запомненное в стеке содержимое регистра (E)SP.

### А.6.2 Арифметические команды

Таблица А.5 – Команды целочисленной арифметики

ADD r/m, r/m/i	Сложение двух операндов: $r/m \leftarrow (r/m + r/m/i)$
XADD r/m, reg	Обмен и сложение (486+)
ADC r/m, r/m/i	Сложение двух операндов с учетом переноса от предыдущей операции: $r/m \leftarrow (r/m + r/m/i + CF)$
INC r/m	Увеличение на 1: $r/m \leftarrow (r/m + 1)$
SUB r/m, r/m/i	Вычитание: $r/m \leftarrow (r/m - r/m/i)$
SBB r/m, r/m/i	Вычитание с заемом: $r/m \leftarrow (r/m - r/m/i - CF)$
DEC r/m	Уменьшение на 1: $r/m \leftarrow (r/m - 1)$
CMP r/m, r/m/i	Сравнение – вычитание без сохранения результата (только установка флагов)
CMPXCHG r/m, reg	Сравнение и обмен данными (486+)
CMPXCHGB	Сравнение и обмен 8 байт (PENTIUM+)

NEG r/m	Изменение знака операнда (преобразование в дополнительном коде): $r/m \leftarrow (0 - r/m)$
MUL r/m	Умножение AL/AX/EAX на беззнаковое целое значение из r/m
IMUL r/m	Умножение AL/AX/EAX на целое знаковое значение из r/m
IMUL reg16/32, r/m	Знаковое умножение reg16/32 на r/m (помещение результата без расширения разрядности в reg16/32) (16 бит – 286+; 32 бита – 386+)
IMUL reg16/32, r/m, immed	Знаковое умножение r/m на 16/32-битовый непосредственный операнд и помещение результата без расширения разрядности в reg16/32 (16 бит – 286+; 32 бита – 386+)
DIV r/m	Деление расширенного аккумулятора на беззнаковое число из r/m
IDIV r/m	Знаковое деление расширенного аккумулятора на знаковое целое из r/m
CBW	Знаковое расширение байта в аккумуляторе (AL) до слова: AH ← заполняется битом AL[7]
CWD	Преобразование слова в двойное слово (расширение знака AX в DX) DX ← заполняется битом AX[15]
CWDE	EAX [16...31] ← заполняется битом AX [15]
CDQ	Преобразование двойного слова в – четверенное: EDX ← заполняется битом EAX [31]
DAA	Коррекция AL после BCD-сложения
DAS	Коррекция AL после BCD-вычитания
AAA	Коррекция AL после ASCII-сложения
AAS	Коррекция AL после ASCII-вычитания
AAM	Коррекция AL после ASCII-умножения
AAD	Коррекция AL, AH перед ASCII-делением

Различие между знаковыми и беззнаковыми числами при выполнении арифметических операций заключается в интерпретации двоичных наборов. Беззнаковые числа – это обычные двоичные числа (все биты значащие), а знаковые числа представлены в дополнительном коде.

Операции сложения и вычитания одинаковы для обоих типов чисел. Единственное отличие заключается в механизме обнаружения выхода за диапазон. Команды сложения и вычитания устанавливают флаг CF, если результат, интерпретируемый как беззнаковое число, оказывается вне диапазона; они же устанавливают флаг OF, если результат, интерпретируемый как знаковое число, выходит за диапазон.

Команда **XADD** – обмена и сложения – обменивает операнды и складывает их. Поэтому на месте операнда-источника остается операнд-получатель, а на месте операнда-получателя формируется сумма.

Команда **NEG** изменяет знак операнда в дополнительном коде.



Команда **CMR** (сравнение) аналогична команде вычитания, но результат нигде не запоминается. Эта команда выставляет флаги, по которым можно определить отношение между двумя операндами: равенство, больше или меньше (см. табл. 6). После команды **CMR** обычно используется команда условного перехода.

Команда **CMRCHG** – сравнение и обмена – воспринимает 3 операнда: операнд-источник в регистре, операнд-получатель в памяти и аккумулятор **AL/AX/EAX**. Если значения в операнде-получателе и аккумуляторе равны, операнд-получатель заменяется операндом-источником. В противном случае исходное значение операнда-получателя загружается в аккумулятор. Флаги отражают результат, полученный при вычитании операнда-получателя из аккумулятора.

Таблица А.6 – Состояние флагов после команды сравнения

Отношение	Знаковые числа	Беззнаковые числа
(dest) > (src)	(ZF=0) & (SF=OF)	(CF=0) & (ZF=0)
(dest) => (src)	SF = OF	CF = 0
(dest) = (src)	ZF = 1	ZF = 1
(dest) <= (src)	(ZF=1) & (SF<>OF)	(CF=1) & (ZF=1)
(dest) < (src)	SF <> OF	CF = 1

Команды умножения могут иметь: одно-, двух- или трехадресную форму.

В одноадресных командах **MUL** и **IMUL** один из сомножителей по умолчанию размещается в аккумуляторе (см. табл. А.7), а второй сомножитель указан в команде. Результат умножения в два раза длиннее операндов.

При двухадресной форме (**IMUL reg16/32,r/m**) или трехадресной форме (**IMUL reg16/32, r/m, immmed**) команд умножения со знаком – результат размещается в регистре-приемнике. В этом случае старшие 16 (или 32) разряда произведения при умножении 16- (или 32-) разрядных операндов теряются. Такие команды удобно применять для вычисления адресов элементов массивов.

Команды деления **DIV** и **IDIV** имеют только одноадресную форму, причем разрядность делимого (табл. А.8) должна вдвое превышать разрядность делителя, указанного в команде.

Знак остатка при выполнении команды **IDIV** устанавливается равным знаку делимого.

Для подготовки операнда-делимого двойной длины используются команды расширения аккумулятора знаковыми битами. При выполнении команд – **CBW / CWDE** (преобразование байта в слово / преобразование слова в двойное слово с расширением в аккумуляторе) – расширенный операнд остается в аккумуляторе. Команды – **CWD / CDQ** (преобразование слова в двойное слово / преобразование двойного слова в четверенное слово) – расширяют аккумулятор **AX** или **EAX** в регистры **DX** или **EDX**

соответственно, куда заносится старшая половина (расширенный знак) операнда.

Таблица А.7 – Размещение первого множителя и результата умножения

Разрядность операндов	Множитель	Результат	
		ст. часть	мл. часть
8	AL	AH	AL
16	AX	DX	AX
32	EAX	EDX	EAX

Таблица А.8 – Размещение делимого и результатов деления

Разрядность делителя	Делимое		Частное	Остаток
	ст. разряды	мл. разряды		
8	AH	AL	AL	AH
16	DX	AX	AX	DX
32	EDX	EAX	EAX	EDX

Система команд процессоров x86 позволяет выполнять арифметические действия над числами, представленными в ДВОИЧНО-ДЕСЯТИЧНОМ УПАКОВАННОМ ФОРМАТЕ (**B CD код**) или в коде ASCII, используемом при обмене информацией и при вводе с клавиатуры. Для этих чисел допустимы значения от 0 до 9 в младшей тетраде.

Команда **DAA** – ДЕСЯТИЧНОЙ КОРРЕКЦИИ АККУМУЛЯТОРА ПОСЛЕ СЛОЖЕНИЯ BCD-чисел выполняет действия над содержимым AL следующим образом:

если содержимое младшей тетрады AL больше 9 или установлен флаг AF = 1, то к содержимому AL добавляется 6 ;

если после этого содержимое старшей тетрады AL стало больше 9 или установлен флаг CF, то число 6 добавляется к старшей тетраде AL.

Аналогичным образом выполняются действия над содержимым AL командой **DAS** – ДЕСЯТИЧНАЯ КОРРЕКЦИЯ ПОСЛЕ ВЫЧИТАНИЯ BCD-чисел:

– если младшая тетрада больше 9 или установлен флаг AF = 1, то из AL вычитается число 6;

– если после этого старшая тетрада больше 9 или установлен флаг CF = 1, то число 6 вычитается из старшей тетрады AL.

Перед выполнением арифметических команд над числами в коде ASCII необходимо очистить старшие тетрады этих чисел. Такие числа называются: распакованными (неупакованными).

Команда **AAA** выполняет коррекцию числа в регистре AL, полученного в результате сложения двух распакованных десятичных операндов. Если содержимое младшей тетрады AL больше 9 или установлен флаг AF = 1, то к содержимому AL добавляется 6; после этого к AH прибавляется 1, очищается старшая тетрада AL, и устанавливаются флаги CF и AF.

Команда **AAS** выполняет коррекцию числа в регистре AL, полученного в результате вычитания двух распакованных десятичных операндов. Если содержимое младшей тетрады AL больше 9 или установлен флаг AF = 1, то из AL вычитается число 6; после этого из AH вычитается 1, очищается старшая тетрада AL, и устанавливаются флаги CF и AF.

Команда **AAM** выполняет коррекцию числа в регистре AL, полученного после умножения двух распакованных десятичных операндов. Содержимое AL делится на 10; частное пересылается в AH, а остаток – в AL.

Команда **AAD** производит коррекцию делимого **ДО ВЫПОЛНЕНИЯ** команды деления. Для этого содержимое регистра AH умножается на 10 и результат прибавляется к содержимому в AL, старший байт аккумулятора AH очищается. Полученный операнд используется для обычного деления на распакованный делитель.

### А.6.3 Логические команды

**Логические двухоперандные команды** служат для реализации трех булевых функций (результат помещается на место первого операнда):

AND – поразрядное логическое И;

OR – поразрядное логическое ИЛИ;

XOR – поразрядное логическое ИСКЛЮЧАЮЩЕЕ ИЛИ (сумма по модулю 2).

Сюда также относится команда TEST (проверка), которая выполняет поразрядное логическое И, но результат никуда не заносит, а только устанавливаются флаги для выполнения условных переходов.

Команды XOR и SUB позволяют обнулить все биты регистра (регистр должен быть и источником и приемником).

Таблица А.9 – Команды логических операций

AND r/m, r/m/i	Побитовое логическое И
TEST r/m, r/m/i	Проверка бит (логическое И без записи результата – установка флагов)
OR r/m, r/m/i	Побитовое логическое ИЛИ
XOR r/m, r/m/i	Побитовое логическое ИСКЛЮЧАЮЩЕЕ ИЛИ
NOT r/m	Побитовая инверсия

### А.6.4 Команды сдвигов

**Команды сдвигов и циклических сдвигов** (табл. А.10) выполняют сдвиг 8/16/32-битового операнда на 1 бит или на произвольное число бит (но не больше длины операнда). Для сдвигов более, чем на один бит, число сдвигов может быть записано предварительно в регистр CL или задано непосредственным операндом в команде (286+). Во всех командах сдвигов последний выдвигаемый бит помещается во флаг CF.

В командах двойного сдвига операндом-приемником (dest) может быть содержимое reg16/32 или mem16/32, операндом-источником (src) – только содержимое РОНа (с разрядностью 16/32). Для сдвигов более, чем на один бит, число сдвигов может быть записано предварительно в регистр CL или задано непосредственным операндом в команде.



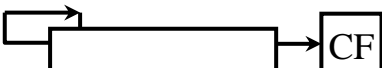
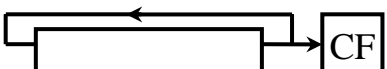
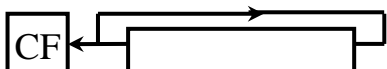
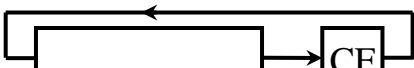
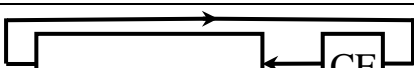
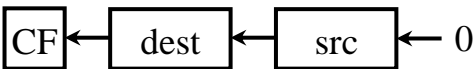
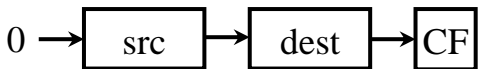
Внутри процессора операнды dest и src объединяются в промежуточном регистре двойной длины, содержимое которого логически сдвигается влево или вправо. После сдвига в операнд-приемник (dest) помещаются соответствующие сдвинутые биты промежуточного регистра. Содержимое операнда-источника (src) не изменяется. Можно сказать, что в этих командах сдвигается операнд-приемник (dest) и в его освобождающиеся биты «вдвигается» содержимое операнда-источника (src).

#### А.6.5 Команды битовых операций

Отсутствуют в МП 86/286.

Команда – **BT r/m,im8** – или – **BT r/m,reg** – (тестирование бита) выбирает из адресуемого регистра или памяти (r/m) значение определенного бита и копирует его во флаг CF. Номер бита (индекс) определяется значением байта непосредственного операнда или задается содержимым регистра (reg).

Таблица А.10 – Команды сдвигов

Команда	Мнемоника	Выполнение команды
Логический сдвиг влево Арифметический сдвиг влево	SHL SAL	
Логический сдвиг вправо	SHR	
Арифметический сдвиг вправо	SAR	
Циклический сдвиг вправо	ROR	
Циклический сдвиг влево	ROL	
Циклический сдвиг вправо через флаг CF	RCR	
Циклический сдвиг влево через флаг CF	RCL	
Двойной сдвиг влево (386+)	SHLD	
Двойной сдвиг вправо (386+)	SHRD	

Когда номер бита (индекс) определен как константа (immed), его диапазон составляет от 0 до 31. Если поле r/m определяет ячейку памяти (размером слово или двойное слово), а номер бита задан содержимым регистра reg, то этот номер бита (индекс) считается целым знаковым числом в диапазоне от -32К до +(32К-1) для 16-битовой операции или от -2Г до +(2Г-1) для 32-битовой операции.

Таблица А.11 – Команды битовых операций (386+)

BT r/m, im8 BT r/m, reg	Тестирование бита – загрузка в CF бита с номером (индексом) im8 из r/m. Загрузка в CF бита из r/m с номером из «reg»
BTC r/m, im8 BTC r/m, reg	Тестирование (загрузка в CF) и инверсия бита
BTR r/m, im8 BTR r/m, reg	Тестирование (загрузка в CF) и сброс бита
BTS r/m, im8 BTS r/m, reg	Тестирование (загрузка в CF) и установка в 1 бита
BSF(BSR) reg, r/m	Сканирование бит вперед (назад) в ячейке r/m. В reg загружается индекс первого единичного бита в ячейке r/m.

Аналогичная команда – **BTS** – после копирования устанавливает адресуемый бит в 1. Команда – **BTR** – после копирования сбрасывает бит, а команда – **BTC** – инвертирует.

Команды – **BSF** и **BSR** – производят сканирование содержимого регистра или ячейки памяти (r/m) и заносят в регистр-приемник (reg) номер первого встреченного единичного бита. При выполнении команды – **BSF** – сканирование начинается с младшего разряда, а в команде – **BSR** – со старшего разряда. Если операнд равен нулю (единичные биты отсутствуют), то устанавливается флаг ZF = 1. При этом содержимое регистра-приемника будет неопределенным. Если единичный бит найден, то флаг ZF = 0.

#### А.6.6 Команды обработки цепочек

Под **ЦЕПОЧКОЙ (строкой)** понимается последовательность байт, слов или двойных слов в памяти, а **ЦЕПОЧЕЧНОЙ (строковой) ОПЕРАЦИЕЙ** называется операция, которая выполняется над каждым элементом цепочки. Например, цепочечная передача производит пересылку целой цепочки из одной области памяти в другую. Сокращение времени выполнения цепочечных команд достигается за счет мощного набора примитивных команд, выполняющих ускоренную обработку каждого элемента цепочки и необходимые служебные действия (табл. 12).

Перед выполнением цепочечных команд необходимо:

– загрузить начальный (конечный) адрес цепочки-источника в регистры DS:(E)SI (допускается замена сегмента) (имеются соответствующие команды: LDS и др.);

– загрузить начальный (конечный) адрес цепочки-приемника в регистры ES:(E)DI (командой LES);

– сбросить флаг DF=0 (командой CLD), если цепочки обрабатываются по возрастанию адресов, или установить флаг DF=1 (командой STD), если цепочки обрабатываются по убыванию адресов;

– при использовании префикса повторения REP в регистр (E)CX загрузить количество повторений цепочечной операции;

– при работе с портами в регистр DX загрузить адрес порта.

Таблица А.12 – Примитивы цепочечных (строковых) команд

MOVS	mem(DI) ← mem(SI),	Модифицировать SI, DI
CMPS	mem(SI) – mem(DI), FLAGS,	Модифицировать SI, DI
SCAS	A – mem(DI), FLAGS,	Модифицировать DI
LODS	A ← mem(SI),	Модифицировать SI
STOS	mem(DI) ← A,	Модифицировать DI
INS	mem(DI) ← port(DX),	Модифицировать DI (286+)
OUTS	port(DX) ← mem(SI),	Модифицировать SI (286+)

Цепочечный примитив **MOVSB (MOVSW, MOVSD)** – передать элемент цепочки – пересылает байт (слово или двойное слово) из ячейки памяти, смещение которой находится в регистре (E)SI (подразумевается, что цепочка-источник по умолчанию находится в текущем сегменте данных, определяемом регистром DS, но допускается замена сегмента), в ячейку памяти со смещением из (E)DI (цепочка-получатель должна находиться только в сегменте, определяемом регистром ES).

При выполнении цепочечной команды содержимое регистров (E)SI и (E)DI автоматически модифицируется так, чтобы адресовать следующие элементы цепочек. Флаг DF определяет автоинкремент (DF = 0) или автодекремент (DF = 1) индексных регистров. Величина инкремента/декремента зависит от размера элементов и составляет 1, 2 или 4, когда элементами цепочек являются, соответственно, байты, слова или двойные слова.

Если в цепочечную команду добавить префикс повторения: **REP MOVSB**, то примитив MOVSB, будет повторяться с уменьшением (E)CX на 1 (после выполнения примитива) до обнуления (E)CX.

Команда сравнения цепочек **CMPSB (CMPSW, CMPSD)** – производит вычитание байта (слова или двойного слова) цепочки приемника (dest) из соответствующего элемента цепочки-источника (src). В зависимости от результата вычитания устанавливаются флаги (в регистре (E)FLAGS), но сами операнды не изменяются. Индексные регистры-указатели продвигаются на следующие элементы цепочек.

Когда перед командой **CMPS** указан префикс повторения **REPE** или **REPZ**), операция интерпретируется как: «сравнивать, пока не достигнут конец цепочек или пока не найден равный элемент».

При наличии префикса **REPNE** (или **REPNZ**) операция приобретает смысл: «сравнивать, пока не достигнут конец цепочек или пока элементы остаются равными».

Команда сканирования цепочек **SCASB** (**SCASW**, **SCASD**) – производит вычитание элемента цепочки (байт, слово или двойное слово) из содержимого аккумулятора **AL/AX/EAX**. В зависимости от результатов вычитания устанавливаются флаги, но значения операндов не изменяется.

С префиксом **REPE** (или **REPZ**) команду **SCAS** можно использовать для поиска элемента цепочки со значением, отличающимся от заданного в аккумуляторе значения. Префикс **REPNE** (или **REPNZ**) позволяет найти элемент цепочки, значение которого равно значению в аккумуляторе.

Команда **LODSB** (**LODSW**, **LOSD**) загружает в аккумулятор (**AL/AX/EAX**) элемент из цепочки (байт, слово или двойное слово) и продвигает указатель (**E**)**SI** на следующий элемент. Обычно эта команда с префиксом повторения не используется.

Команда сохранения аккумулятора в цепочке **STOSB** (**STOSW**, **STOSD**) – передает байт (слово или двойное слово) из аккумулятора **AL/AX/EAX** в элемент цепочки и продвигает регистр-указатель (**E**)**DI** на следующий элемент. С префиксом повторения **REP** эта команда удобна для инициализации цепочки на фиксированное значение.

Команды ввода и вывода цепочек **INSB** (**INSW**, **INSD**) и **OUTSB** (**OUNSW**, **OUNSD**) как и обычные команды ввода/вывода являются привилегированными.

Команда **INS** вводит данные из порта, адресуемого регистром **DX**, в ячейку памяти с адресом **ES:(E)DI**. После ввода операнда производится модификация регистра (**E**)**DI** на 1, 2 или 4 с учетом состояния флага направления **DF**.

Команда **OUTS** выводит данные из ячейки памяти с адресом **DS:(E)SI** в выходной порт, адрес которого находится в регистре **DX**. После вывода операнда производится коррекция указателя (**E**)**SI**.

Обе эти команды могут использоваться с префиксом повторения **REP**. В этом случае ввод или вывод данных повторяется до обнуления регистра-счетчика (**E**)**CX**.

Необходимо отметить, что пять мнемоник префикса повторения **REP**, **REPE/REPZ**, **REPNE/REPNZ** определяют только два объектных (машинных) кода префикса (**0F2h** и **0F3h**), а пять мнемоник введены для лучшей передачи содержательного смысла задачи.

#### А.6.7 Команды работы с флагами (флажковые команды)

Однобайтовые команды этой группы позволяют модифицировать некоторые флаги регистра (**E**)**FLAGS** (см. табл. А.13). Остальные флаги

могут быть модифицированы после записи содержимого флагового регистра в регистр или ячейку памяти (например, командой PUSHF(D)), с последующим возвратом во флаговый регистр.

Команды, модифицирующие флаг IF, являются IOPL-чувствительными, т.е. выполняющая их программа должна иметь текущий уровень привилегий CPL, меньший или равный содержимому поля IOPL в регистре (E)FLAGS. Если это условие не выполняется, возникает нарушение общей защиты.

Таблица А.13 – Команды работы с флагами

CLC	$CF \leftarrow 0$	Сброс флага переноса
CMC	$CF \leftarrow 1 - CF$	Инверсия флага переноса
STC	$CF \leftarrow 1$	Установка флага переноса
CLD	$DF \leftarrow 0$	Сброс флага направления цепочек DF
STD	$DF \leftarrow 1$	Установка флага направления DF
CLI	$IF \leftarrow 0$	Запрет маскируемых аппаратных прерываний
STI	$IF \leftarrow 1$	Разрешение маскируемых аппаратных прерываний
CTS (CLTS)	$TF \leftarrow 0$	Сброс флага переключения задач
LAHF	Загрузка младшего байта регистра флагов в AH	
SAHF	Сохранение AH в младшем байте регистра флагов	

#### А.6.8 Команды передачи управления

КОМАНДА БЕЗУСЛОВНОГО ПЕРЕХОДА с общей мнемоникой **JMP** имеет 5 форм, различающихся расстоянием до адреса назначения от текущей команды и способом задания назначения (целевого адреса – target).

В коротком (SHORT) внутрисегментном переходе двухбайтовая команда JMP rel8 содержит во втором байте смещение в дополнительном коде (максимально возможный переход: назад – 128 или вперед +127 от адреса команды, находящейся после команды JMP).

Команда прямого внутрисегментного перехода (NEAR) аналогична предыдущей, но полное смещение в дополнительном коде содержит 16 (или 32 бита), которое прибавляется к текущему значению (E)IP. Эта форма команды передает управление в любую точку текущего сегмента кода.

В команде косвенного внутрисегментного перехода JMP r/m адрес целевого назначения (target) загружается в (E)IP из регистра или ячейки памяти.

Команда прямого межсегментного перехода JMP prt содержит непосредственный операнд, содержащий: 16-битовый селектор, который загружается в регистр CS, и 16- (или 32-) битовое смещение, загружаемое в (E)IP.



Команда косвенного межсегментного перехода адресует в памяти полный 32- (или 48-) битовый указатель – селектор: смещение. Селектор загружается в регистр CS, а смещение – в регистр (E)IP.

КОМАНДЫ УСЛОВНЫХ ПЕРЕХОДОВ (табл. А.14) осуществляют передачу управления в зависимости от результатов предыдущих операций. Все команды условных переходов производят передачу управления только в пределах текущего сегмента кода (т.е. содержимое сегментного регистра CS не изменяется), если заданное в команде условие удовлетворяется. Переход реализуется прибавлением находящегося в команде смещения (в дополнительном коде) к содержимому регистра (E)IP. В процессорах 86/286 8-ми битовое смещение обеспечивает диапазон перехода от – 128 до +127 байт. В процессорах 386+ наряду с таким смещением допускается также полное 16- или 32-битовое смещение в дополнительном коде. Этим обеспечивается переход в любую точку текущего сегмента кода.

В мнемокодах команд условных переходов при сравнении чисел со знаком используются буквы:

– G (greater) – больше,

– L (less) – меньше.

Для чисел без знака:

– A (above) – над, выше,

– B (below) – под, ниже.

Условие равенства:

– E (equal) – равно;

Невыполнение некоторого условия:

– N (not) – не.

Таблица А.14 – Команды передачи управления (переходов)

JMP target	Безусловный переход к целевому адресу target
J(E)CXZ target	Условный переход, если (E)CX = 0
LOOP target	Декремент (E)CX и переход, если (E)CX <> 0
LOOPE target (LOOPZ) target	Декремент (E)CX и переход, если (E)CX <> 0 & ZF = 1
LOOPNE target (LOOPNZ) target	Декремент (E)CX и переход, если (E)CX <> 0 & ZF = 0
Jccc target	Команды условного перехода
CALL target	Вызов процедуры (подпрограммы)
RET (n)	Возврат из процедуры. Необязательный параметр n задает коррекцию значения указателя стека
SETccc r/m	Условное заполнение байта. Если выполняется условие «ccc», все биты байта dest (регистра или памяти) устанавливаются в 1, иначе – в 0. Условия «ccc» те же, что и в командах условных переходов (386+)

Для некоторых команд условных переходов зарезервированы два или три альтернативных мнемозкода (см. табл. А.15), подчеркивающих содержательный смысл проверяемого условия.

Таблица А.15 – Кодирование условий перехода

Код поля ссс	Мнемоника поля ссс	Состояние флагов	Условие перехода
0000	O	OF=1	Переполнение
0001	NO	OF=0	Не переполнение
0010	B/NAE/C	CF=1	Ниже / не выше или равно
0011	AE/NB/NC	CF=0	Не ниже / выше или равно
0100	E/Z	ZF=1	Равно / нуль
0101	NE/NZ	ZF=0	Не равно / не нуль
0110	BE/NA	CF=1 & ZF=1	Ниже или равно / не выше
0111	NBE/A	CF=0 & ZF=0	Не ниже или равно / выше
1000	S	SF=1	Есть знак (отрицательный)
1001	NS	SF=0	Нет знака (положительный)
1010	P/PE	PF=1	Есть паритет / четный паритет
1011	NP/PO	PF=0	Нет паритета / нечетный паритет
1100	L/NGE	ZF<>OF	Меньше / не больше или равно
1101	NL/GE	SF=OF	Не меньше / больше или равно
1110	LE/NG	(SF<>OF) & ZF=1	Меньше или равно / не больше
1111	NLE/G	SF=(OF & ZF)	Не меньше или равно / больше

КОМАНДЫ ВЫЗОВА ПОДПРОГРАММЫ (процедуры) CALL передает управление с автоматическим сохранением в стеке адреса возврата (текущего содержимого IP), т.е. адреса команды, находящейся после команды CALL. В конце подпрограммы последняя команда RET восстанавливает из стека в регистр IP адрес возврата.

Команда CALL имеет такие же формы (относительную, прямую и косвенную), как и команда JMP; отсутствует только короткая (SHORT) форма. По воздействию на регистры CS и (E)IP команда CALL также соответствует команде JMP, но дополнительно включает в текущий сегмент стека адрес возврата с соответствующей коррекцией указателя стека (E)SP.

Команда RET допускает указание в поле операнда непосредственной константы `immed16`. В таких командах после извлечения из стека адреса возврата константа `immed16` прибавляется к содержимому регистра (E)SP. В результате в стеке пропускаются параметры, переданные подпрограмме.

Команда заполнения байта по условию (SETссс r8/m8) (см. табл. 14) предназначена для того, чтобы сохранить зафиксированное флагами условие для дальнейших вычислений. Мнемоника условия «ссс» полностью совпадает с условием переходов (табл. А.15).

КОМАНДЫ ПРЕРЫВАНИЯ. Двухбайтовая команда INT n (табл. А.16) в начале включает в стек содержимое регистра флагов (E)FLAGS и полный

адрес возврата, представленный содержимым регистров CS и (E)IP. Кроме этого сбрасывается в нуль флаг разрешения прерываний IF. После этого осуществляется косвенный переход через элемент «n» дескрипторной таблицы прерываний IDT.

Однобайтовый вариант этой команды INT 3 называется прерыванием контрольной точки.

Команда прерывания INTO эквивалентна команде INT 4, если установлен флаг переполнения OF = 1. Когда же флаг OF = 0, команда INTO не производит никаких действий.

Команда возврата из прерывания IRET извлекает из стека сохраненные в нем адрес возврата и регистр флагов.

Таблица А.16 – Команды прерывания

INT n	Выполнение программного прерывания
INT 3	Однобайтовая команда прерывания по типу 3
INTO	Выполнение программного прерывания 4, если OF=1
IRET	Возврат из прерывания

## ПРИЛОЖЕНИЕ Б

### АРХИТЕКТУРНОЕ РАСШИРЕНИЕ 32-РАЗРЯДНЫХ МИКРОПРОЦЕССОРОВ – ПРОЦЕССОР ЧИСЕЛ С ПЛАВАЮЩЕЙ ТОЧКОЙ (FPU)

#### Б.1 Форматы чисел с плавающей точкой

Сопроцессор x87 распознает три формата чисел с плавающей точкой, хранимых в памяти (рис. Б.1). Внутри сопроцессора все числа преобразуются в расширенный формат.

Каждый формат состоит из трех полей: знак (S), порядок и мантисса (рис. Б.1). Числа в этих форматах занимают в памяти: 4, 8 или 10 байт.

Байт с наименьшим адресом в памяти является младшим байтом мантиссы. Байт с наибольшим адресом содержит семь старших бит порядка и **БИТ ЗНАКА (S)**. Знак кодируется: 0 – плюс, 1 – минус.

**В ПОЛЕ МАНТИССЫ** хранятся только *нормализованные числа*. Для этого необходимо скорректировать порядки (т.е. сдвинуть запятую) так, чтобы в целой части числа (в двоичной системе счисления) до запятой **была 1**. Поэтому все мантиссы представляются в форме:

$$1.XXXXXXXX...XXX, \quad \text{где: } X = 0 \text{ или } 1$$

Одинарная точность (Короткое вещественное)	1 бит	8 бит	23 бита	4 байта
	S	Порядок	Мантисса	
Двойная точность (Длинное вещественное)	1 бит	11 бит	52 бита	8 байт
	S	Порядок	Мантисса	
Расширенная точность (Временное вещественное)	1 бит	15 бит	64 бита	10 байт
	S	Порядок	Мантисса	

Рис. Б.1 – Форматы чисел с плавающей точкой

Но если старший бит всегда содержит 1, ее можно не хранить в каждом числе с плавающей точкой. Поэтому ради дополнительного бита точности сопроцессор x87 хранит числа одинарной и двойной точности **без старшего бита мантиссы** (с неявным старшим битом). Исключением является кодирование нуля – нулевые поля мантиссы и порядка.

Числа с расширенной точностью хранятся и обрабатываются **с явным старшим битом**.

**ПОЛЕ ПОРЯДКА** определяет степень числа 2, на которую нужно умножить нормализованную мантиссу для получения исходного значения числа с плавающей точкой.

Чтобы хранить *отрицательные порядки*, в ПОЛЕ ПОРЯДКА находится сумма истинного порядка и положительной константы, называемой СМЕЩЕНИЕМ. Для одинарной точности смещение равно 127, для двойной точности 1023, для расширенной точности 16383 (т.е. половине максимального порядка). Двоичный код смещения для всех порядков равен : 0111...111.

Числа в поле порядка: 00000..00 и 11111..111 – зарезервированы для спецкодирования или обработки ошибок.

Запись чисел с плавающей точкой в памяти:

Одинарная точность:  $(-1)^S (1.X_1 X_2 \dots X_{23}) * 2^{(E-127)}$ ,

Двойная точность:  $(-1)^S (1.X_1 X_2 \dots X_{52}) * 2^{(E-1023)}$ ,

Расширенная точность:  $(-1)^S (X_1.X_2 \dots X_{64}) * 2^{(E-16383)}$ ,

где  $S$  – значение знакового бита;

$X_1 X_2 \dots$  – биты, хранимые в поле мантиссы;

$E$  – число, хранимое в поле порядка.

Расширенный формат используется преимущественно внутри сопроцессора для представления промежуточных результатов, чтобы упростить получение округленных окончательных результатов в формате двойной точности.

Диапазон представления чисел с плавающей точкой в десятичной системе счисления приведен в табл. Б.1 и рис. Б.2.

Таблица Б.1 – Диапазон представления чисел в десятичной системе

Формат	Значащих десятичных цифр	Наименьшая степень числа 10	Наибольшая степень числа 10
Одинарный	7	-37	38
Двойной	15	-307	308
Расширенный	19	-4931	4932

Имеется несколько особых случаев представления вещественных чисел:

– *нуль* – число, у которого порядок и мантисса равны нулю; может иметь положительный или отрицательный знаки, которые в операциях сравнения игнорируются;

– *наименьшее положительное и наибольшее отрицательное* числа – имеют значение порядка, равное 1, и значение мантиссы, равное 0; отличаются знаковыми разрядами;

– *наибольшее положительное и наименьшее отрицательное* числа – имеют поле порядка, в котором все биты кроме самого младшего, равны 1, и содержат единицы во всех разрядах мантиссы; отличаются знаковыми разрядами;

– *положительная и отрицательная бесконечность* – число, которое содержит все единицы в поле порядка и все нули в поле мантиссы; отличаются знаковыми разрядами;

– *нечисло (NaN)* содержит все единицы в поле порядка и любое значение в поле мантиссы; может возникнуть в результате выполнения неправильной операции при замаскированных особых случаях;

– *неопределенность* – в поле порядка содержатся все единицы, а в поле мантиссы – число 1000...0 (для одинарной и двойной точности) или 11000...0 (для расширенной точности).

Если результат арифметической операции меньше наименьшего отрицательного числа или больше наибольшего положительного числа конкретного формата, то говорят, что операция вызвала ПЕРЕПОЛНЕНИЕ. Если же результат арифметической операции ненулевой, но находится между наибольшим отрицательным и наименьшим положительным числами конкретного формата, то говорят, что в операции возникло АНТИПЕРЕПОЛНЕНИЕ.

Отметим, что рис. Б.2 абсолютно симметричен для положительных и отрицательных чисел. Следовательно, операция нахождения абсолютного значения никогда не может вызвать ни переполнения, ни антипереполнения.

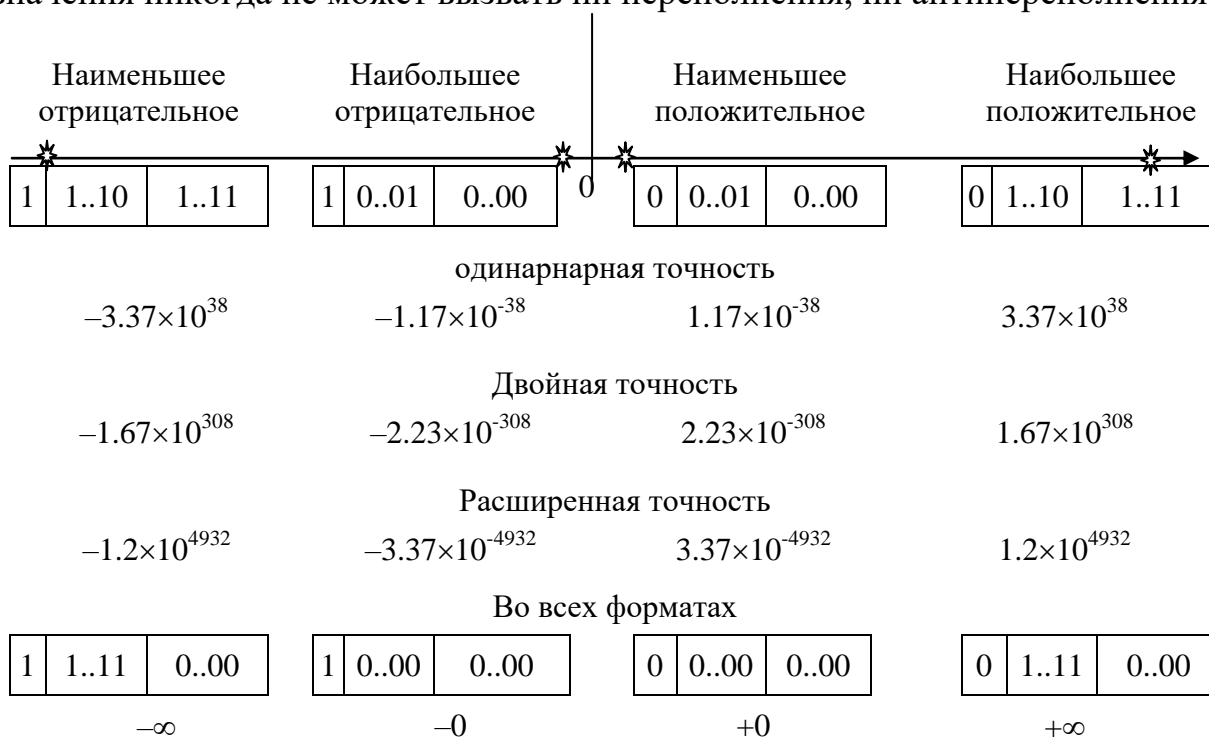


Рис. Б.2 – Диапазон представления чисел

Сопроцессор x87 имеет команды, которые преобразуют целые числа в числа с плавающей точкой и – наоборот. Это необходимо при вычислениях, в которых имеются числа обоих форматов. Допустимые форматы **ЦЕЛЫХ ЧИСЕЛ** приведены на рис. Б.3.

Целое число	Дополнительный код	16 бит, 2 байта
Короткое целое	Дополнительный код	32 бита, 4 байта
Длинное целое	Дополнительный код	64 бита, 8 байт
Упакованное десятичное	<div> <div>8 бит</div> <div>72 бита</div> </div> <div> <div>S</div> <div>0000000</div> <div>18 десятичных тетрад</div> </div>	10 байт

Рис Б.3 – Форматы целых чисел

**ЦЕЛОЕ СЛОВО** – это 16-битовое целое число процессора x86 в **дополнительном коде**. **КОРОТКОЕ ЦЕЛОЕ** и **ДЛИННОЕ ЦЕЛОЕ** похожи на целое слово, но их длина больше.

**УПАКОВАННОЕ ДЕСЯТИЧНОЕ ЧИСЛО** состоит из 18 десятичных цифр, размещенных по две в байте. Знаковый бит находится в дополнительном 10-м байте. Младшие семь бит этого байта должны быть нулевыми.

При выполнении арифметических операций над числами с плавающей точкой иногда возникают ошибочные условия или **ОСОБЫЕ СЛУЧАИ**, описанные далее.

**НЕДЕЙСТВИТЕЛЬНАЯ ОПЕРАЦИЯ** – включает в себя, например, деление и умножение с операндами бесконечность и нуль, извлечение корня квадратного из отрицательного числа, попытку использовать несуществующий регистр сопроцессора x87 или др.

**ДЕНОРМАЛИЗОВАННЫЙ ОПЕРАНД** – возникает, когда ради увеличения диапазона приходится жертвовать точностью.

**ДЕЛЕНИЕ НА НУЛЬ** – дает в результате бесконечность. Наличие у сопроцессора x87 двух нулей (см. рис. Б.16) очевидным образом приводит к знаковым бесконечностям:

$$\begin{aligned}
 x / (+0) &= +\infty, & -x / (+0) &= -\infty, \\
 x / (-0) &= -\infty, & -x / (-0) &= +\infty.
 \end{aligned}$$

Сопроцессоры 87/287 имеет два режима управления бесконечностью: **ПРОЕКТИВНЫЙ** и **АФФИННЫЙ**.

В **ПРОЕКТИВНОМ РЕЖИМЕ** факт наличия двух бесконечностей скрыт (как скрыт факт наличия двух нулей), т.е. положительная бесконечность замыкается на отрицательную бесконечность (сравнение двух бесконечностей всегда дает ответ "равны"). Режим проективной бесконечности удобен для вычисления рациональных функций (значения в полюсах можно представить бесконечностями) и цепных дробей.

В **АФФИННОМ РЕЖИМЕ** распознаются две бесконечности (см. рис. Б.2) и два нуля; здесь все конечные числа «х» удовлетворяют условию:

$$-\infty < x < +\infty.$$

Аффинный режим либеральнее проективного, поскольку допускает больше операций над бесконечностями.

В сопроцессорах 387+ (и 287 XL) оставлено только аффинное представление бесконечности.

**ЧИСЛЕННОЕ ПЕРЕПОЛНЕНИЕ** – возникает, когда результат слишком велик по абсолютной величине, чтобы быть представленным в формате приемника.

**ИСЛЕННОЕ АНТИПЕРЕПОЛНЕНИЕ** – возникает, когда ненулевой результат по абсолютной величине слишком мал для представления, т.е. когда он слишком близок к нулю.

**НЕТОЧНЫЙ РЕЗУЛЬТАТ** – возникает, когда результат операции невозможно точно представить в формате приемника. Например, при делении 1.0 на 3.0 получается бесконечная дробь, которую невозможно точно представить ни в одном формате. Если особый случай неточного результата замаскирован, сопроцессор x87 округляет результат до обычного числа с плавающей точкой. Программист может выбирать один из четырех режимов округления (результаты операций на рис. 2.9 представлены звездочками).

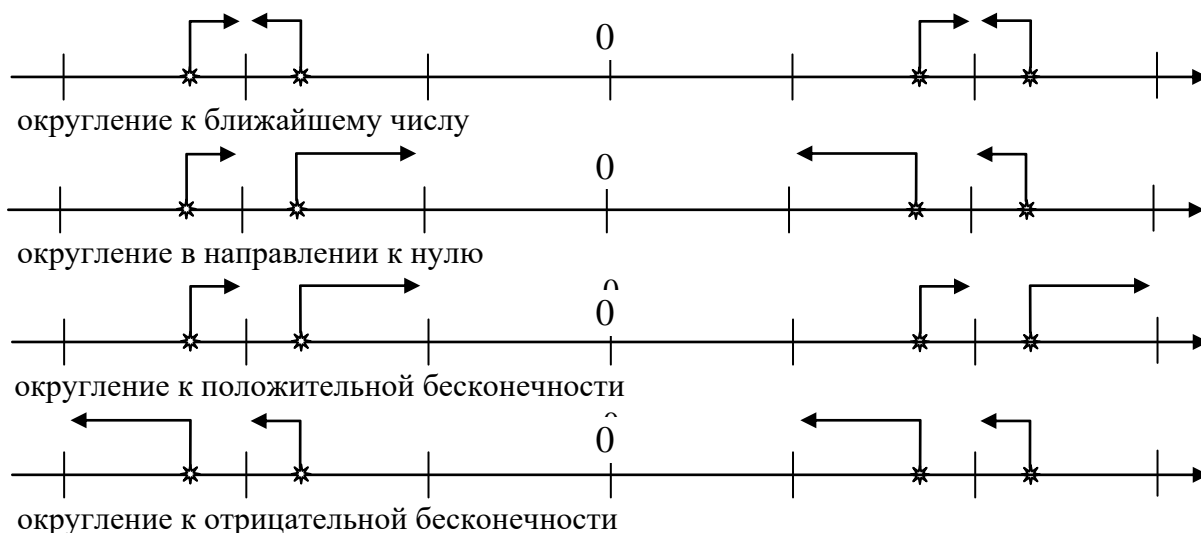


Рис. Б.4 – Режимы округления неточного результата

## Б.2 Регистры сопроцессора x87

Сопроцессор x87 имеет регистры (рис. Б.5), удобные для вычислений над числами с плавающей точкой.

Операнды команд сопроцессора x87 могут находиться в памяти или в одном из восьми **ЧИСЛЕННЫХ РЕГИСТРОВ**. Эти регистры хранят числа преимущественно в формате расширенной точности. Обращение к операндам в регистрах осуществляется намного быстрее, чем к операндам в памяти.

Номер численного регистра, указанного в команде, сопроцессор всегда суммирует с содержимым поля **TOP** (или **ST** – вершина стека) в регистре состояния. Сумма (берущаяся по модулю 8) определяет используемый



численный регистр, т. е. регистры адресуются внутри сопроцессора, как замкнутый в кольцо стек (рис. Б.6).

16-битовый **РЕГИСТР УПРАВЛЕНИЯ** содержит поле МАСОК ОСОБЫХ СЛУЧАЕВ и поля округления чисел (рис. Б.7).

Младшие 6 битов отведены для МАСОК ОСОБЫХ СЛУЧАЕВ:

- IM – маска недействительной операции,
- DM – маска денормализованного операнда,
- ZM – маска деления на нуль,
- OM – маска переполнения,
- UM – маска антипереполнения,
- PM – маска точности (неточного результата).

Численные регистры

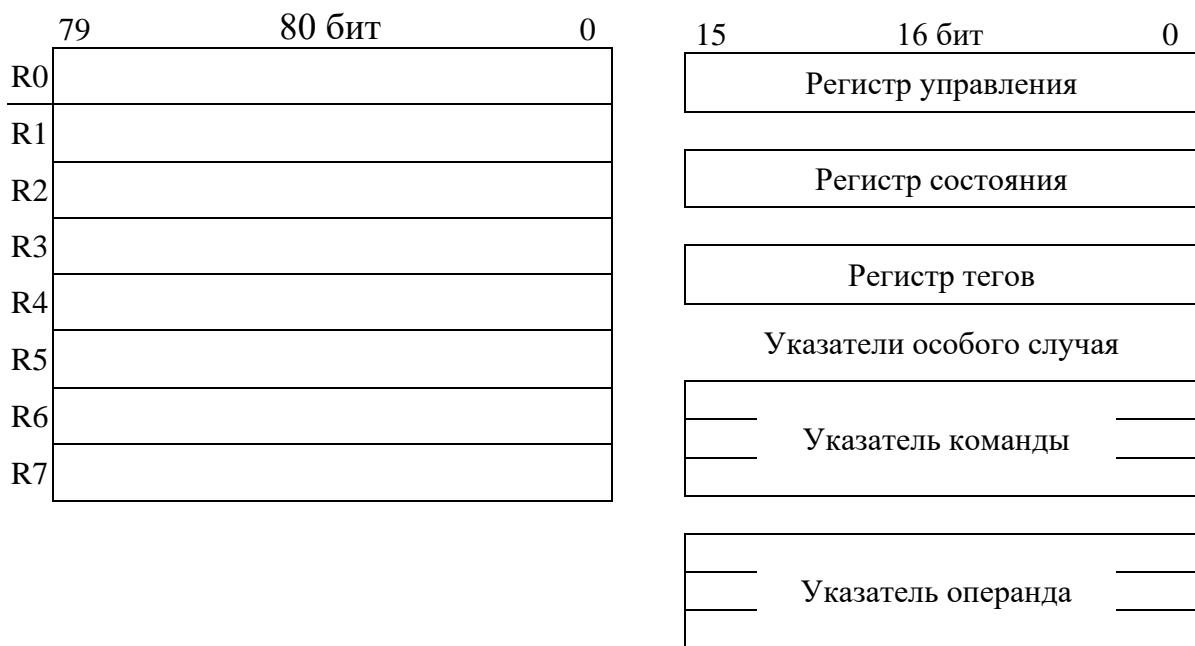


Рис. Б.5 – Регистры сопроцессора x87

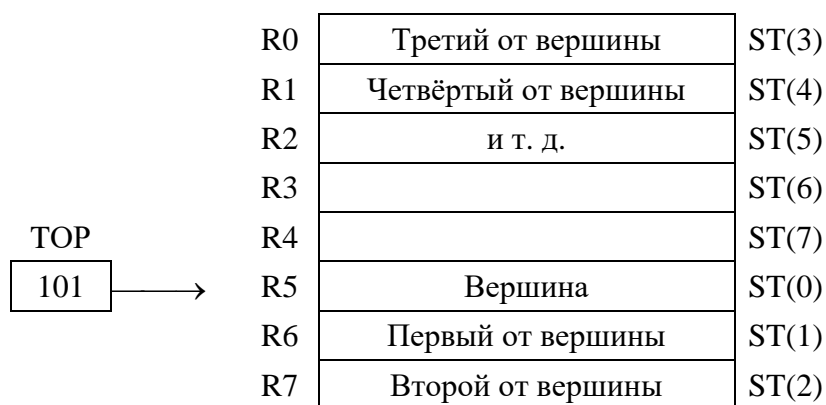


Рис. Б.6 – Адресация численных регистров как стека

16-битовый **РЕГИСТР СОСТОЯНИЯ** содержит флаги, модифицируемые после выполнения команд, а также поле вершины стека (TOP) (см. рис. Б.7).

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SW	B	C3		TOP		C2	C1	C0	ES	SF	PE	UE	OE	ZE	DE	IE

Рис. Б.7 – Регистр состояния (Status Word)

Младшие 6 бит содержат ФЛАГИ ОСОБЫХ СЛУЧАЕВ:

IE – недействительная операция,

DE – денормализованный операнд,

ZE – деление на нуль,

OE – переполнение,

UE – антипереполнение,

PE – точность (неточный результат).

При возникновении численного особого случая (замаскированного или нет) сопроцессор устанавливает соответствующий флаг в 1.

Флаги особых случаев «зависают», т.е. сбросить их в нуль должен программист, загружая в регистр состояния новое значение.

Бит НАРУШЕНИЯ СТЕКА – SF – устанавливается в 1, если команда вызывает переполнение стека (включение в уже заполненный стек) или антипереполнение стека (извлечение из пустого стека). Когда SF = 1, бит кода условия C1 показывает переполнение (C1 = 1) или антипереполнение (C1 = 0) стека.

Бит СУММАРНОЙ ОШИБКИ – ES – устанавливается в 1, когда команда порождает любой незамаскированный особый случай.

Биты C0, C1, C2, C3 содержат КОДЫ УСЛОВИЙ, являющиеся результатом сравнения или команды нахождения остатка. Интерпретация кода условия зависит от конкретной команды.

Поле ВЕРШИНЫ СТЕКА – TOP (Stack Top) – содержит номер регистра, являющегося верхним элементом стека (рис. Б.6). Его содержимое прибавляется (по модулю 8) ко всем номерам численных регистров.

Бит ЗАНЯТОСТИ – B – устанавливается в 1, когда сопроцессор 8087 выполняет команду или сигнализирует прерывание, а когда сопроцессор свободен, этот бит сбрасывается в 0.

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

Рис. Б.8 – Регистр управления (Control Word)

16-битовый **РЕГИСТР УПРАВЛЕНИЯ** содержит маски особых случаев. Когда бит маски равен 0, возникновение соответствующего особого случая вызовет приостановку программы и появление прерывания процессора x86. Если же бит маски установлен в 1, то соответствующий особый случай замаскирован и формируются специальные значения чисел.

Два бита (8-й и 9-й) поля **УПРАВЛЕНИЯ ТОЧНОСТЬЮ** – PC – заставляют сопроцессор x87 округлять все числа перед загрузкой их в численные регистры до указанной точности:

- 11 – округление до расширенной точности (принимается по умолчанию),
- 10 – округление до двойной точности,
- 00 – округление до одинарной точности.

Задание пониженной точности (сокращение длины мантиссы) ликвидирует достоинство формата PT (расширенной точности), но не повышает быстродействие сопроцессора.

Два бита (10-й и 11-й) поля **УПРАВЛЕНИЯ ОКРУГЛЕНИЕМ** – RC – выбирают один из четырех режимов округления (см. рис. Б.4):

- 00 – округление к ближайшему (принимается по умолчанию),
- 01 – округление к отрицательной бесконечности,
- 10 – округление к положительной бесконечности,
- 11 – округление к нулю.

Бит **УПРАВЛЕНИЯ БЕСКОНЕЧНОСТЬЮ** – IC – задает режим интерпретации бесконечности:

- 0 – проективный режим (принимается по умолчанию),
- 1 – аффинный режим.

В сопроцессорах 387+ бит 12 слова управления игнорируется. Используется только аффинный режим.

16-битовый **РЕГИСТР ТЭГОВ** состоит из 8 двухбитовых полей (рис. Б.9). Каждое поле соответствует своему численному регистру и индицирует состояние регистра:

- 00 – действительное число (т.е. любое конечное ненулевое число),
- 01 – нуль,
- 10 – недействительное число (например, нечисло или бесконечность),
- 11 – пустой регистр.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Тэг7	Тэг6	Тэг5	Тэг4	Тэг3	Тэг2	Тэг1	Тэг0								

Рис. Б.9 – Регистр тэгов

Регистры отмечаются как пустые, если они не инициализированы или их содержимое было «извлечено» из стека численных регистров. Сопроцессор (FPU) использует этот тэг для обнаружения антипереполнения стека (слишком много извлечений) и переполнения стека (слишком много

включений). Обе ситуации приводят к возникновению особого случая недействительной операции. Когда этот особый случай замаскирован, но в операции возникает переполнение или антипереполнение стека, сопроцессор корректирует ST, как будто ничего необычного не произошло, и возвращает как результат операции неопределенность.

**УКАЗАТЕЛИ ОСОБОГО СЛУЧАЯ** (команды и операнда) (32 или 48 битов) предназначены для процедур обработки особых случаев. Они имеют два формата – в зависимости от работы сопроцессора в реальном или виртуальном режимах (рис. Б.10).

Реальный режим		Виртуальный режим	
Адрес команды (0..15)		Смещение команды	
Адрес команды (16..19)	Код операции (0..10)	Селектор команды	
Адрес операнда (0..15)		Смещение операнда	
Адрес операнда (16..19)	-----	Селектор операнда	

Рис. Б.10 – Указатели особого случая

В виртуальном режиме сопроцессор выдает селектор и смещение подозрительной команды и ее операнда в памяти (если он есть).

В реальном режиме указатели содержат 20-битовые адреса этих объектов, а также 11 младших бит кода операции.

### Б.3 Команды сопроцессора x87

Форматы команд сопроцессора x87 аналогичны форматам команд x86. Ассемблерные мнемоники команд сопроцессора начинаются с буквы **F** (Floating) и их легко обнаружить, т.к. таких мнемоник у процессора x86 нет.

Вторая буква **I** (Integer) обозначает операцию с целым двоичным числом из памяти, буква **B** (Binari-coded decimal) – операцию с десятичным операндом из памяти, а вторая "пустая" буква определяет операцию с вещественными числами (с плавающей точкой).

Предпоследняя или последняя буква **R** (reveres) указывает обратную операцию (для вычитания и деления).

Последняя буква **P** (Poring) идентифицирует команду, заключительным действием которой является извлечение из стека.

В командах неявно указывается численный регистр сопроцессора, адресуемый вершиной стека ST или ST(0) (Stack Top).

Численные регистры адресуются относительно вершины стека. Например, команда:

FADD ST(0), ST(3)

прибавляет содержимое третьего регистра от вершины стека к содержимому верхнего регистра стека.

Верхний регистр стека можно обозначить ST или ST(0). Например, команда:

FADD ST(0), ST(0)

прибавляет содержимое верхнего регистра стека к нему же, т.е. удваивает содержимое регистра ST(0).

Для операндов в памяти допускаются все режимы адресации процессора x86, например:

FADD [BX]  
FADD ANAME [BX] [SI]

**В командах FPU НЕ ИСПОЛЬЗУЕТСЯ НЕПОСРЕДСТВЕННАЯ АДРЕСАЦИЯ** операндов.

### Б.3.1 Команды передачи данных сопроцессора x87

Команды включения в стек FLD, FILD, FBLD и все команды включения констант – сначала уменьшают указатель стека на 1, преобразуют операнд-источник в расширенный формат (если он уже не представлен в таком формате) и помещают его в новую вершину стека.

Мнемоники команд с целочисленным (двоичным) операндом начинаются с «FI», а с десятичным операндом – с «FB».

Таблица Б.2 – Команды передачи данных

Мнемоника, Операнд			Тип команды
С плав. точкой	Целое число	Десят.число	
FLD FSTP FST FXCH	FILD FISTP FIST -	FBLD FBSTP - -	(-) Включить в стек Извлечь из стека (+) Копирование Обмен регистров
FLDZ FLD1 FLDPI FLDLG2 FLDLN2 FLDL2T FLDL2E	(-) Включить в стек 0 (-) Включить в стек 1 (-) Включить в стек $= 3.1415...$ (-) Включить в стек $\log_{10}(2)$ (-) Включить в стек $\ln(2)$ (-) Включить в стек $\log_2(10)$ (-) Включить в стек $\log_2(e)$		

Обозначения: (–) – декремент указателя стека до включения операнда в вершину стека; (+) – инкремент указателя стека после извлечения операнда из вершины стека.

Команды извлечения из стека преобразуют содержимое вершины стека в необходимый формат, помещают его в операнд-приемник (память или регистр) и после этого инкрементируют указатель стека.

Команды копирования делают то же самое, но не изменяют указатель стека. Отсутствующую команду FBST можно реализовать двумя командами:

FLD ST(0) ; продублировать вершину стека с декрементом  
FBSTP ; извлечь из стека с инкрементом указателя.

При выполнении команд передачи данных может возникнуть необходимость указать новую вершину стека. Команды FINCSTP и FDECSTP осуществляют соответственно инкремент и декремент указателя стека ST. Они не влияют на регистр тэгов и численные регистры.

### Б.3.2 Арифметические команды

Базовые арифметические команды – сложение, вычитание умножение и деление – имеют два операнда (источник и приемник) и реализуют действия:

ПРИЕМНИК ← ПРИЕМНИК \$ ИСТОЧНИК,  
где: \$ - основные команды : +, –, \*, / .

Для некоммутативных команд вычитания и деления имеются обратные варианты команд (в конце мнемоники добавляется буква R – reverses):

ПРИЕМНИК ← ИСТОЧНИК \$ ПРИЕМНИК.

**ВО ВСЕХ КОМАНДАХ ОДИН ОПЕРАНД ДОЛЖЕН БЫТЬ В ВЕРШИНЕ СТЕКА.** Мнемоники всех базовых арифметических команд приведены в табл. Б.3.

Таблица Б.3 – Мнемоники базовых арифметических команд

Команды	Основная	Целое в памяти	С извлечением
Сложение	FADD	FIADD	FADDP
Вычитание	FSUB	FISUB	FSUBP
Обратное вычитан.	FSUBR	FISUBR	FSUBRP
Умножение	FMUL	FIMUL	FMULP
Деление	FDIV	FIDIV	FDIVP
Обратное деление	FDIVR	FIDIVR	FDIVRP

Имеется 6 форм базовых команд. Действия всех шести форм команд иллюстрируется на примере команды вычитания.

FSUB mem,

FISUB mem – адресуемый операнд в памяти является источником, а регистр вершины стека ST(0) – приемником. Преобразование в расширенный

формат с плавающей точкой осуществляется в процессе выполнения команды. УКАЗАТЕЛЬ СТЕКА НЕ МОДИФИЦИРУЕТСЯ.

FSUB ST, ST(i) – любой численный регистр ST(i) служит источником, а ST(0) – приемником. УКАЗАТЕЛЬ СТЕКА НЕ МОДИФИЦИРУЕТСЯ.

FSUB ST(i), ST – вершина стека является источником, а ST(i) – приемником. УКАЗАТЕЛЬ СТЕКА НЕ МОДИФИЦИРУЕТСЯ.

FSUBP ST(i), ST – вершина стека является источником, а ST(i) – приемником. По окончании операции источник ST(0) извлекается из стека с последующим ИНКРЕМЕНТОМ УКАЗАТЕЛЯ СТЕКА (рис. Б.11).

FSUB – (команда с классической стековой адресацией – использует только ST1, ST0) – извлекает из вершины стека источник (потом инкрементирует указатель стека), затем извлекает приемник (еще раз инкрементирует указатель стека), выполняет операцию и перед включением результата в стек декрементирует указатель. В итоге – ВЕРШИНА СТЕКА СДВИНУЛАСЬ В СТОРОНУ УВЕЛИЧЕНИЯ (рис. Б.12). Последняя форма является частным случаем предыдущей.

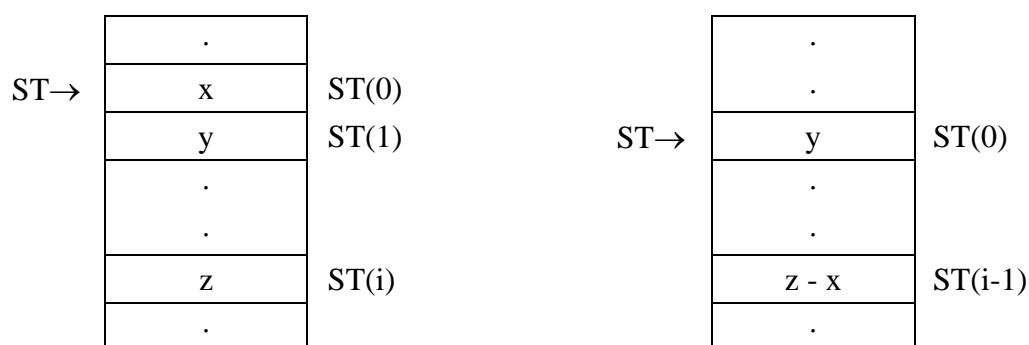


Рис. Б.11 – Действие команды FSUBP ST(i),ST

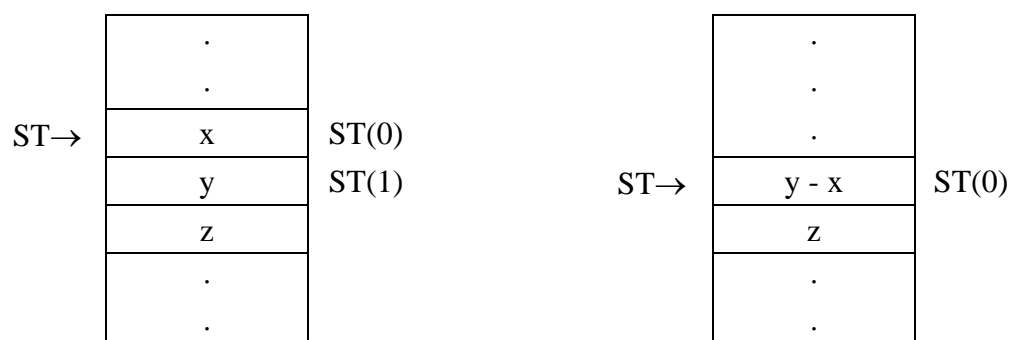


Рис. Б.12 – Действие команды FSUB

### Б.3.3 Дополнительные арифметические команды

Эти команды без явных операндов выполняют действия над содержимым вершины стека, результат помещают туда же БЕЗ МОДИФИКАЦИИ УКАЗАТЕЛЯ СТЕКА.

FABS – нахождение абсолютной величины.

FCHS – изменение знака операнда.

FRNDINT – округление операнда до целого в формате с плавающей точкой.

FSQRT – извлечение квадратного корня.

FPREM – вычисляет остаток от деления содержимого ST(0) на число из ST(1). Остаток замещает число в ST(0).

FSCALE – масштабирование на степень числа 2 – прибавляет целое число из ST(1) к порядку в регистре ST(0), т.е. умножает (или делит) ST(0) на число  $2^{ST(1)}$ . Эту команду можно использовать для возведения числа 2 в целую степень (положительную или отрицательную).

FXTRACT – разлагает содержимое ST(0) на два числа: несмещенный порядок (замещает старое значение в ST(0)) и знаковую мантиссу (включаемую сверху, т.е. в ST(7)).

Команда FSCALE, находящаяся после команды FXTRACT, восстанавливает исходное число.

### Б.3.4 Команды сравнения

FCOM ST(i)/mem – сравнивает содержимое ST(0) с операндом в численном регистре или в памяти, т.е. производит вычитание операндов без запоминания результата и устанавливает коды условий в регистре состояния (таблица Б.4).

FICOM mem – сравнивает содержимое вершины стека ST(0) с целым числом в памяти.

FCOMP ST(i)/mem – аналогична команде FCOM, но после сравнения производит извлечение операнда из вершины стека.

FCOMPP ST(i) – сравнивает ST(0) с ST(i) и извлекает из стека оба операнда.

FTST – сравнивает вершину стека с нулем.

FXAM – сравнивает вершину стека с нулем, но выставляет 4 флага условий (в частности, определяется ненормализованная мантисса, бесконечность, нечисло и др.).

FCOMI ST(0),ST(i) – сравнение вещественных чисел и установка флагов в EFLAGS (P6+).

Таблица Б.4 – Коды условий после сравнения

C3	C0	Условие
0	0	ST(0) > x
0	1	ST(0) < x
1	0	ST(0) = x
1	1	ST(0) и x – не сравнимы



FCOMIP ST(0),ST(i) – сравнение вещественных чисел и установка флагов в EFLAGS и извлечение операнда из вершины стека (P6+).

Флаги условий (C0, C3) сопроцессора x87 используются для организации условных переходов микропроцессором x86. Для этого командой – FSTSW AX – содержимое регистра состояния x87 копируется в аккумулятор AX микропроцессора x86. После этого командой – SAHF – старший байт аккумулятора (AH) передается в младший байт регистра флагов. При этом условию C0 соответствует флаг CF, а условию C3 – флаг ZF.

### Б.3.5 Трансцендентные команды

К элементарным трансцендентным функциям относятся:

- тригонометрические функции (sin, cos, tg и др.),
- обратные тригонометрические функции (arcsin, arctg и др.),
- логарифмические функции ( $\log_2(x)$ ,  $\log_{10}(x)$ ,  $\log_e(x)$ ),
- показательные функции ( $y^x$ ,  $2^x$ ,  $10^x$ ,  $e^x$ ),
- гиперболические функции (sh, ch, th и др.),
- обратные гиперболические функции (arsh, arch, arth и др.).

Таблица Б.5 – Трансцендентные команды

Мнемоника	Описание команды	Вычисляемая функция
FPTAN	Частичный тангенс	$ST(1) / ST(0) = \text{tg} (ST(0))$
FSIN	Синус(387+)	$ST(0) = \sin (ST(0))$
FCOS	Косинус (387+)	$ST(0) = \cos (ST(0))$
FSINCOS	Синус, косинус (387+)	$ST(7) = \sin (ST(0));$ $ST(0) = \cos (ST(0))$
FPATAN	Частичный арктангенс	$ST(0) = \text{arctg} (ST(1)/ST(0))$
FYL2X	Двоичный логарифм	$ST(0) = ST(1) * \log_2 (ST(0))$
FYL2XP1	Двоичный логарифм	$ST(0) = ST(1) * \log_2 (ST(0)+1)$
F2XM1	Показательная функция	$ST(0) = 2^{(ST(0))} - 1$

Сопроцессор x87 вычисляет любую из элементарных трансцендентных функций от аргументов двойной точности, давая результат двойной точности с ошибкой младшего разряда округления.

Команда **FPTAN** нахождения частичного тангенса в качестве результата выдает два числа (сопроцессоры 87/287):

$$y / x = \text{tg} (ST(0)).$$

Число «у» заменяет старое содержимое ST(0), а число «х» включается сверху. Поэтому, после выполнения команды указатель стека уменьшится на 1, число «х» будет записано в новую вершину стека ST(0), а число «у» – в регистр ST(1).

Для получения значения тангенса необходимо выполнить команду FDIV. Две команды FPTAN и FDIV выбирают аргумент из вершины стека и туда же помещают значение **тангенса** (БЕЗ МОДИФИКАЦИИ УКАЗАТЕЛЯ ВЕРШИНЫ СТЕКА). Две команды FPTAN и FDIVR вычисляют значение **котангенса**.

Для команды FPTAN аргумент задается в радианах и должен находиться в диапазоне (сопроцессоры 87/287):

$$0 \leq ST(0) \leq 1/4.$$

Для СОПРОЦЕССОРОВ 387+ аргумент команды FPTAN (в радианах) может быть любым:

$$-2^{63} \leq ST(0) \leq +2^{64}.$$

Значение тангенса исходного угла  $tg(ST(0))$  замещает аргумент и в стек включается сверху 1,0 (для программной совместимости с предыдущими сопроцессорами 87/287).

Значения остальных тригонометрических функций (для сопроцессоров 87/287) можно вычислить, используя формулы тангенса половинного угла (табл. Б.6). Поэтому перед началом вычисления тригонометрических функций с использованием команды FPTAN необходимо аргумент в  $ST(0)$  поделить на 2. Новое значение аргумента «z» должно также удовлетворять условию:  $0 \leq z \leq 1/4$ .

Таблица Б.6 – Формулы для вычисления тригонометрических функций

$\sin(z) = \frac{2 * (y / x)}{1 + (y / x)^2}$	$\cos(z) = \frac{1 - (y / x)^2}{1 + (y / x)^2}$
$tg(ST(0)) = y / x$	$ctg(ST(0)) = x / y$
$\sec(z) = \frac{1 + (y / x)^2}{2 * (y / x)}$	$\cos ec(z) = \frac{1 + (y / x)^2}{1 - (y / x)^2}$

В СОПРОЦЕССОРАХ 387+ появились новые команды:

**FSIN** – вычисление синуса;

**FCOS** – вычисление косинуса;

**FSINCOS** – вычисление синуса и косинуса.

Все они воспринимают в  $ST(0)$  исходный угол, **измеряемый в радианах** и находящийся в диапазоне:  $-2^{63} \leq ST(0) \leq +2^{63}$ . Команды FSIN и FCOS возвращают результат на место аргумента, а команда FSINCOS возвращает значение синуса на место аргумента и включает значение косинуса в стек.

Команда **FPATAN** вычисляет  $\arctg (ST(1)/ST(0))$ . Два операнда извлекаются из стека, а результат включается в стек. Поэтому окончательно, **УКАЗАТЕЛЬ СТЕКА УВЕЛИЧИВАЕТСЯ НА 1**. Операнды этой команды для сопроцессоров 8087/287 должны удовлетворять условию:

$$0 < ST(1) < ST(0).$$

В сопроцессорах 387+ ограничений на диапазон допустимых аргументов команды **FPATAN** не существует.

Для вычисления остальных обратных тригонометрических функций по аргументу « $z$ » необходимо предварительно подготовить операнды в  $ST(0)$  и  $ST(1)$  в соответствии с табл. Б.7 (делить операнды не нужно).

Таблица Б.7 – Формулы для вычисления обратных тригонометрических функций

$\arcsin(z) = \arctg\left(\frac{z}{\sqrt{(1-z^2)}}\right)$		$\leftarrow ST(1)$ $\leftarrow ST(0)$
$\arccos(z) = 2 * \arctg\left(\frac{\sqrt{(1-z)}}{\sqrt{(1+z)}}\right)$		$\leftarrow ST(1)$ $\leftarrow ST(0)$
$\arctg(z) = \arctg\left(\frac{z}{1}\right)$	$\leftarrow ST(1)$ $\leftarrow ST(0)$	$\arccotg(z) = \arctg\left(\frac{1}{z}\right)$ $\leftarrow ST(1)$ $\leftarrow ST(0)$
$\operatorname{arccosec}(z) = \arctg\left(\frac{\operatorname{sign}(z)}{\sqrt{(z^2-1)}}\right)$		$\leftarrow ST(1)$ $\leftarrow ST(0)$
$\operatorname{arcsec}(z) = 2 * \arctg\left(\frac{\sqrt{( z -1)}}{\sqrt{( z +1)}}\right)$		$\leftarrow ST(1)$ $\leftarrow ST(0)$

Команда **FYL2X** вычисляет функцию:  $Y = ST(1) * \log_2 ST(0)$ . Два операнда извлекаются из стека, а затем результат включается в стек. Поэтому **УКАЗАТЕЛЬ СТЕКА УВЕЛИЧИТСЯ НА 1**. В команде требуется удовлетворение естественного для логарифмической функции условия:

$$ST(0) > 0.$$

Значения других логарифмических функций вычисляются по формулам в табл. Б.8 с загрузкой в регистр  $ST(1)$  необходимых констант командами: **FLDLN2** и **FLDLG2**.

Таблица Б.8 – Формулы для вычисления логарифмических функций

$\log_2(x) \rightarrow \text{FLD1} ; \text{FLD } x ; \text{FYL2X} ;$
$\ln(x) = \ln(2) * \log_2(x) \rightarrow \text{FLDLN2} ; \text{FLD } x ; \text{FYL2X} ;$
$\lg(x) = \lg(2) * \log_2(x) \rightarrow \text{FLDLG2} ; \text{FLD } x ; \text{FYL2X} .$

Еще одна логарифмическая команда **FYL2XP1** вычисляет функцию:  $Y = ST(1) * \log_2(ST(0) + 1)$ . Причина появления этой команды заключается в получении более высокой точности вычисления функции:  $\log(1 + x)$ . Эта функция часто встречается в финансовых расчетах, а также при вычислении обратных гиперболических функций.

Команда показательной функции **F2XM1** вычисляет:

$$Y = 2^{(ST(0))} - 1.$$

Аргумент показательной функции должен находиться в диапазоне : для сопроцессоров 87/287:

$$0 \leq ST(0) \leq 0.5;$$

для сопроцессоров 387+:

$$-1 \leq ST(0) \leq +1.$$

Вычисление функции  $2^x - 1$  вместо функции  $2^x$  позволяет избежать потери точности, когда аргумент «х» близок к 0 (а значение функции  $2^x$  близко к 1). Остальные показательные функции вычисляются по формулам в табл. Б.9.

Таблица Б.9 – Формулы для вычисления показательных функций

$2^x = [2^x - 1] + 1 = \text{F2XM1}(x) + 1 ;$ $e^x = 2^{(x * \log_2(e))} = \text{F2XM1}(x * \log_2(e)) + 1 ;$ $10^x = 2^{(x * \log_2(10))} = \text{F2XM1}(x * \log_2(10)) + 1 ;$ $a^x = 2^{(x * \log_2(a))} = \text{F2XM1}(x * \log_2(a)) + 1 .$
---

Константы  $\log_2(e)$  и  $\log_2(10)$  уже имеются в сопроцессоре и загружаются соответствующими командами (см. таблицу Б.2)

Таблица Б.10 – Формулы для вычисления гиперболических функций

Синус гиперболический	$sh(x) = \frac{sign(x)}{2} * \left[ \left( e^{ x } - 1 \right) + \frac{e^{ x } - 1}{e^{ x }} \right]$
Косинус гиперболический	$ch(x) = \frac{1}{2} * \left( e^{ x } + \frac{1}{e^{ x }} \right)$
Тангенс гиперболический	$th(x) = sign(x) - \left[ \frac{e^{(2 *  x )} - 1}{e^{(2 *  x )} + 1} \right]$
Котангенс гиперболический	$1 / th(x)$
Косеканс гиперболический	$1 / sh(x)$
Секанс гиперболический	$1 / ch(x)$

Таблица Б.11 – Формулы для вычисления обратных гиперболических функций

$arsh(x) = sign(x) * \ln(2) * \log_2(1 + z),$ где: $z =  x  + \frac{ x }{(1/ x ) + \sqrt{1 + (1/ x )}}$
$arch(x) = \ln(2) * \log_2(1 + z),$ где: $z = x - 1 + \sqrt{(x^2 - 1)}$
$arth(x) = sign(x) * \ln(2) * \log_2(z),$ где: $z = \frac{2 *  x }{1 -  x }$
$archth(x) = arth(1 / x)$
$arcsh(x) = arsh(1 / x)$
$arsch(x) = arch(1 / x)$

### Б.3.6 Команды управления сопроцессором x87

Команды управления сопроцессором x87 обеспечивают доступ к нечисловым регистрам. Мнемоники, которые начинаются с FN, соответствуют командам «БЕЗ ОЖИДАНИЯ», т.е. процессор x86 передает их для выполнения в сопроцессор x87, не проверяя занятость сопроцессора и игнорируя численные особые случаи.

Мнемоники без буквы «N» соответствуют командам «С ОЖИДАНИЕМ», т.е. заставляют процессор x86 реагировать на незамаскированные особые случаи и ожидать завершения выполнения

команд в сопроцессоре x87. В общем случае, программистам рекомендуется избегать форм команд «без ожидания».

Команда – FNSTCW mem (FSTCW mem) – передает содержимое регистра управления (CW) в ячейку памяти.

Команда – FLDCW mem – загружает регистр управления (CW) из ячейки памяти. Эти две команды применяются для изменения режима работы сопроцессора x87.

Команда – FNSTSW mem (FSTSW mem) – передает содержимое регистра состояния (SW) сопроцессора x87 в ячейку памяти.

Команда – FNSTSW AX (FSTSW AX) – передает содержимое регистра состояния (SW) сопроцессора в регистр AX микропроцессора x86.

Команда – FNCLEX (FCLEX) – сбрасывает в регистре состояния сопроцессора флаги особых случаев, а также биты ES и BUSY. Эти флаги не сбрасываются аппаратно и должны явно сбрасываться программистом.

Команда – FNINIT (FINIT) – инициализирует регистры управления, состояния и тэгов на значения, приведенные в табл. Б.12. Такое же действие производит аппаратный сигнал сброса – RESET.

Таблица Б.12 – Инициализация сопроцессора x87

Регистр	Выбор	Режим работы
Регистр управления	(Режим бесконечности)	Проективный – (287) Афинный – (387+)
	Режим округления	Округление к ближайшему
	Точность	Расширенная
	Все особые случаи	Замаскированы
Регистр Состояния	Бит занятости	B = 0: Не занят
	Код условия	Не определен
	Указатель стека	TOP = 000
	Бит суммарной ошибки	ES = 0
Регистр тэгов		Все тэги показывают – "пустой"

Пример. Фрагмент программы для подсчета суммы положительных значений  $\text{tg}(X)$ , при  $X = 1 \dots 25$ . Сумма сохраняется в ячейке Sum в длинном вещественном формате.

```
finit
```

```
;--- Вычислить значения pi/2 и pi/4
fldpi
mov     word ptr [Count],2
fild    dword ptr [Count]
fdiv
fst     qword ptr [Pi2]
fild    dword ptr [Count]
fdiv
```

```

        fstp    qword ptr [Pi4]

        mov     CX,25
        mov     word ptr [Count],0

Retry:
        inc     word ptr [Count] ; увеличить X
        call    Tangens ; вычислить тангенс
        fldz
        fcomp   ; результат < 0?
        fstsw   Status
        fwait
        mov     AH,byte ptr [Status+1]
        sahf
        jnb     Noadd ; да
        fld     qword ptr [Sum] ; наращивание суммы
        fadd
        fst     qword ptr [Sum]

Noadd:
        finit
        loop retry

; . . .

Tangens proc near

        push    CX
        fld     dword ptr [Count] ; включить X в стек
        fldpi   ; включить Pi в стек
        fxch    ; обменять X и Pi
        ftst    ; проверить знак X
        xor     BX,BX ; флаг обратной величины
        xor     CX,CX ; флаг изменения знака
        fstsw   Status ; извлечь SW
        fwait
        mov     AH, byte ptr [Status+1] ; перенести SW в p-p флагов
        sahf
        jae     Norm_1 ; X > 0
        fchs    ; изменить знак
        not     CX

;--- приведение X в диапазон [0...Pi]
Norm_1:
        fprem   ; вычислить остаток R от
        fstsw   Status ; деления X на Pi
        fwait
        mov     AH,byte ptr [Status + 1]
        sahf
        jp      Norm_1

;--- приведение X в диапазон [0...Pi/2]
Norm_2:
        fld     qword ptr [Pi2] ; включить Pi/2 в стек

```

```

fstp    ST(2)      ; R → ST(0), Pi → ST(1)
fcom     ; сравнить R с Pi/2
fstsw   Status
fwait
mov     AH,byte ptr [Status + 1]
sahf
jbe     Norm_3     ; R < Pi/2
fsub    ST,ST(1)   ; вычесть Pi/2 из R
not     BX
not     CX

```

;--- приведение X в диапазон [0...Pi/4]

Norm\_3:

```

fld     qword ptr [Pi4] ; включить Pi/4 в стек
fcom     ; сравнить R с Pi/4
fstsw   Status
fwait
mov     AH,byte ptr [Status + 1]
sahf
jae     Tang
fsubr   ST,ST(1) ; вычесть Pi/4 из R
not     BX

```

Tang:

```

fptan           ; найти частичный тангенс
and     CX,CX   ; нужно изменить знак?
jz      Noneg   ; нет
fchs

```

Noneg:

```

and     BX,BX   ; нужно найти обратную величину?
jz      Nobr    ; нет
fxch

```

Nobr:

```

fdiv           ; вычислить тангенс
pop      CX
ret

```

Tangens endp

```

Count    dd 0
Sum       dq 0
Temp     dq 0
Status   dw 0
Pi2      dq 0
Pi4      dq 0

```



## ПРИЛОЖЕНИЕ В

### ЗНАЧЕНИЯ СИГНАТУРЫ ИДЕНТИФИКАЦИИ CPU X86-64

Таблица В.1 – Поля сигнатуры процессоров i486+ Signatures (поле Extended Family имеет значение 00000000b для всех приведенных CPU)

Extended Model	Type	Family Code	Model No	Stepping ID	Description
0000	00	0100	000x	xxxx <sup>(1)</sup>	i486 DX
0000	00	0100	0010	xxxx <sup>(1)</sup>	i486 SX
0000	00	0100	0011	xxxx <sup>(1)</sup>	i487
0000	00	0100	0011	xxxx <sup>(1)</sup>	i486 DX2
0000	00	0100	0011	xxxx <sup>(1)</sup>	i486 DX2 OverDrive
0000	00	0100	0100	xxxx <sup>(3)</sup>	i486 SL
0000	00	0100	0101	xxxx <sup>(1)</sup>	i486 SX2
0000	00	0100	0111	xxxx <sup>(3)</sup>	Write-Back Enhanced i486 DX2
0000	00	0100	1000	xxxx <sup>(3)</sup>	i486 DX4
0000	0x	0100	1000	xxxx <sup>(3)</sup>	i486 DX4 OverDrive
0000	00	0101	0001	xxxx <sup>(2)</sup>	Pentium (60, 66)
0000	00	0101	0010	xxxx <sup>(2)</sup>	Pentium (75, 90, 100, 120, 133, 150, 166, 200)
0000	01 <sup>(4)</sup>	0101	0001	xxxx <sup>(2)</sup>	Pentium OverDrive for Pentium (60, 66)
0000	01 <sup>(4)</sup>	0101	0010	xxxx <sup>(2)</sup>	Pentium OverDrive for Pentium (75, 90, 100, 120, 133)
0000	01	0101	0011	xxxx <sup>(2)</sup>	Pentium OverDrive for i486
0000	00	0101	0100	xxxx <sup>(2)</sup>	Pentium with MMX technology (166, 200)
0000	01	0101	0100	xxxx <sup>(2)</sup>	Pentium OverDrive with MMX technology for Pentium (75, 90, 100, 120, 133)
0000	00	0110	0001	xxxx <sup>(2)</sup>	Pentium Pro
0000	00	0110	0011	xxxx <sup>(2)</sup>	Pentium II
0000	01	0110	0011	xxxx <sup>(2)</sup>	Pentium II OverDrive
0000	00	0110	0101 <sup>(5)</sup>	xxxx <sup>(2)</sup>	Pentium II, Pentium II Xeon, Intel Celeron
0001	00	0110	0101	xxxx <sup>(2)</sup>	Intel EP80579 Integrated Processor, Intel EP80579 Integrated Processor with Intel QuickAssist Technology
0000	00	0110	0110	xxxx <sup>(2)</sup>	Celeron
0001	00	0110	0110	xxxx <sup>(2)</sup>	Celeron (65 nm).
0000	00	0110	0111 <sup>(6)</sup>	xxxx <sup>(2)</sup>	Pentium III, Pentium III Xeon
0000	00	0110	1000 <sup>(7)</sup>	xxxx <sup>(2)</sup>	Pentium III, Pentium III Xeon, Celeron
0000	00	0110	1001	xxxx <sup>(2)</sup>	Pentium M, Celeron M
0000	00	0110	1010	xxxx <sup>(2)</sup>	Pentium III Xeon
0000	00	0110	1011	xxxx <sup>(2)</sup>	Pentium III
0000	00	0110	1101	xxxx <sup>(2)</sup>	Pentium M, Celeron M (90 nm).
0000	00	0110	1110	xxxx <sup>(2)</sup>	Core Duo, Core Solo (65 nm).
0000	00	0110	1111	xxxx <sup>(2)</sup>	Core2 Duo, Core2 Duo mobile, Core2 Quad, Core2 Quad mobile, Core2 Extreme, Pentium Dual-Core, Xeon (65 nm).
0001	00	0110	0111	xxxx <sup>(2)</sup>	Core2 Extreme, Xeon (45 nm).
0001	00	0110	1100	xxxx <sup>(2)</sup>	Atom (45 nm).
0001	00	0110	1010	xxxx <sup>(2)</sup>	Core i7, Xeon (45 nm).
0001	00	0110	1101	xxxx <sup>(2)</sup>	Xeon MP (45 nm).
0001	00	0110	1110	xxxx <sup>(2)</sup>	Core i5/i7, Core i5/i7 Mobile, Xeon (45 nm).

## Продолжение таблицы В.1

Extended Model	Type	Family Code	Model No	Stepping ID	Description
0010	00	0110	1110	xxxx <sup>(2)</sup>	Xeon MP (45 nm).
0010	00	0110	1111	xxxx <sup>(2)</sup>	Xeon MP (32 nm).
0010	00	0110	1100	xxxx <sup>(2)</sup>	Core i7, Xeon (32 nm).
0010	00	0110	0101	xxxx <sup>(2)</sup>	Core i3, Core i5/i7 Mobile (32 nm).
0010	00	0110	1010	xxxx <sup>(2)</sup>	2nd Generation Core Family (Mobile and Desktop), Xeon E3-1200 Family (Sandy Bridge - 32 nm).
0010	00	0110	1101	xxxx <sup>(2)</sup>	Xeon E5 Family (Sandy Bridge - 32 nm).
0011	00	0110	1010	xxxx <sup>(2)</sup>	3rd Generation Core Family (Mobile and Desktop), Xeon E3-1200 v2 Family (Sandy Bridge - 22 nm).
0000	00	1111	0000	xxxx <sup>(2)</sup>	Pentium 4, Xeon (0.18 micron).
0000	00	1111	0001	xxxx <sup>(2)</sup>	Pentium 4, Xeon, Xeon MP, Celeron (0.18 micron).
0000	00	1111	0010	xxxx <sup>(2)</sup>	Pentium 4, Mobile Pentium 4 – M, Xeon, Xeon MP, Celeron, Mobile Celeron (0.13 micron).
0000	00	1111	0011	xxxx <sup>(2)</sup>	Pentium 4, Xeon, Celeron D (90 nm).
0000	00	1111	0100	xxxx <sup>(2)</sup>	Pentium 4, Pentium 4 Extreme Edition, Pentium D, Xeon, Xeon MP, Celeron D (90 nm).
0000	00	1111	0110	xxxx <sup>(2)</sup>	Pentium 4, Pentium D, Pentium Extreme Edition, Xeon, Xeon MP, Celeron D (65 nm).

## Notes:

1. This processor does not implement the CPUID instruction.
2. Refer to corresponding processor Specification Update for the latest list of stepping numbers.
3. Stepping 3 implements the CPUID instruction.
4. The definition of the type field for the OverDrive processor is 01h. An erratum on the Pentium OverDrive processor will always return 00h as the type.
5. To differentiate between the Pentium II processor, model 5, Pentium II Xeon processor and the Celeron processor, model 5, software should check the cache descriptor values through executing CPUID instruction with EAX = 2. If no L2 cache is returned, the processor is identified as an Celeron processor, model 5. If 1M or 2M L2 cache size is reported, the processor is the Pentium II Xeon processor otherwise it is a Pentium II processor, model 5 or a Pentium II Xeon processor with 512-KB L2 cache size.
6. To differentiate between the Pentium III processor, model 7 and the Pentium III Xeon processor, model 7, software should check the cache descriptor values through executing CPUID instruction with EAX = 2. If 1M or 2M L2 cache size is reported, the processor is the Pentium III Xeon processor otherwise it is a Pentium III processor or a Pentium III Xeon processor with 512-KB L2 cache size.
7. To differentiate between the Pentium III processor, model 8 and the Pentium III Xeon processor, model 8, software should check the Brand ID values through executing CPUID instruction with EAX = 1.
8. To differentiate between the processors with the same processor Vendor ID, software should execute the Brand String functions and parse the Brand String.

## ПРИЛОЖЕНИЕ Г

### ФЛАГИ СВОЙСТВ ПРОЦЕССОРОВ X86-64

Таблица Г.1 – Флаги свойств, возвращаемые в регистре EDX

Bit	Name	Description (when Flag = 1)	Comments
0	FPU	Floating-point Unit On-Chip	The processor contains an FPU that supports the Intel387 floating-point instruction set.
1	VME	Virtual Mode Extension	The processor supports extensions to virtual-8086 mode.
2	DE	Debugging Extension	The processor supports I/O breakpoints, including the CR4.DE bit for enabling debug extensions and optional trapping of access to the DR4 and DR5 registers.
3	PSE	Page Size Extension	The processor supports 4-MB pages.
4	TSC	Time Stamp Counter	The RDTSC instruction is supported including the CR4.TSD bit for access/privilege control.
5	MSR	Model Specific Registers	Model Specific Registers are implemented with the RDMSR, WRMSR instructions
6	PAE	Physical Address Extension	Physical addresses greater than 32 bits are supported.
7	MCE	Machine-Check Exception	Machine-Check Exception, INT18, and the CR4.MCE enable bit are supported.
8	CX8	CMPXCHG8 Instruction	The compare and exchange 8-bytes instruction is supported.
9	APIC	On-chip APIC Hardware	The processor contains a software-accessible local APIC.
10		Reserved	-
11	SEP	Fast System Call	Indicates whether the processor supports the Fast System Call instructions, SYSENTER and SYSEXIT (regarding SEP feature bit).
12	MTRR	Memory Type Range Registers	The processor supports the Memory Type Range Registers specifically the MTRR_CAP register.
13	PGE	Page Global Enable	The global bit in the page directory entries (PDEs) and page table entries (PTEs) is supported, indicating TLB entries that are common to different processes and need not be flushed. The CR4.PGE bit controls this feature.
14	MCA	Machine-Check Architecture	The Machine-Check Architecture is supported, specifically the MCG_CAP register.
15	CMOV	Conditional Move Instruction	The processor supports CMOVcc, and if the FPU feature flag (bit 0) is also set, supports the FCMOVCC and FCOMI instructions.
16	PAT	Page Attribute Table	Indicates whether the processor supports the Page Attribute Table. This feature augments the Memory Type Range Registers (MTRRs), allowing an operating system to specify attributes of memory on 4K granularity through a linear address.

## Продолжение таблицы Г.1

Bit	Name	Description (when Flag = 1)	Comments
17	PSE-36	36-bit Page Size Extension	Indicates whether the processor supports 4-MB pages that are capable of addressing physical memory beyond 4-GB. This feature indicates that the upper four bits of the physical address of the 4-MB page is encoded by bits 13-16 of the page directory entry.
18	PSN	Processor serial number is present and enabled	The processor supports the 96-bit processor serial number feature, and the feature is enabled. The Pentium 4 and subsequent processor families do not support this feature.
19	CLFSH	CLFLUSH Instruction	Indicates that the processor supports the CLFLUSH instruction.
20		Reserved	-
21	DS	Debug Store	Indicates that the processor supports the ability to write debug information into a memory resident buffer. This feature is used by the branch trace store (BTS) and precise event-based sampling (PEBS) facilities.
22	ACPI	Thermal Monitor and Software Controlled Clock Facilities	The processor implements internal MSRs that allow processor temperature to be monitored and processor performance to be modulated in predefined duty cycles under software control.
23	MMX	MMX technology	The processor supports the MMX technology instruction set extensions to Intel Architecture.
24	FXSR	FXSAVE and FXSTOR Instructions	The FXSAVE and FXRSTOR instructions are supported for fast save and restore of the floating point context. Presence of this bit also indicates that CR4.OSFXSR is available for an operating system to indicate that it supports the FXSAVE and FXRSTOR instructions.
25	SSE	Streaming SIMD Extensions	The processor supports the Streaming SIMD Extensions to the Intel Architecture.
26	SSE2	Streaming SIMD Extensions 2	Indicates the processor supports the Streaming SIMD Extensions 2 Instructions.
27	SS	Self-Snoop	The processor supports the management of conflicting memory types by performing a snoop of its own cache structure for transactions issued to the bus.
28	HTT	Multi-Threading	The physical processor package is capable of supporting more than one logical processor. This field does not indicate that Hyper-Threading Technology (HTT) or Core Multi-Processing (CMP) has been enabled for this specific processor. To determine if HTT or CMP is supported, compare value returned in EBX[23:16] after executing CPUID with EAX=1. If the resulting value is > 1, then the processor supports Multi-Threading. IF (CPUID(1).EBX[23:16] > 1) { Multi-Threading = TRUE } ELSE { Multi-Threading = FALSE }

Продолжение таблицы Г.1

Bit	Name	Description (when Flag = 1)	Comments
29	TM	Thermal Monitor	The processor implements the Thermal Monitor automatic thermal control circuitry (TCC).
30		Reserved	-
31	PBE	Pending Break Enable	The processor supports the use of the FERR#/ PBE# pin when the processor is in the stop-clock state (STPCLK# is asserted) to signal the processor that an interrupt is pending and that the processor should return to normal operation to handle the interrupt. Bit 10 (PBE enable) in the IA32_MISC_ENABLE MSR enables this capability.

Таблица Г.2 – Флаги свойств, возвращаемые в регистре ECX

Bit	Name	Description (when Flag = 1)	Comments
0	SSE3	Streaming SIMD Extensions 3	The processor supports the Streaming SIMD Extensions 3 instructions.
1	PCLMULDQ	PCLMULDQ Instruction	The processor supports PCLMULDQ instruction.
2	DTES64	64-Bit Debug Store	Indicates that the processor has the ability to write a history of the 64-bit branch to and from addresses into a memory buffer.
3	MONITOR	MONITOR/MWAIT	The processor supports the MONITOR and MWAIT instructions.
4	DS-CPL	CPL Qualified Debug Store	The processor supports the extensions to the Debug Store feature to allow for branch message storage qualified by CPL.
5	VMX	Virtual Machine Extensions	The processor supports Intel Virtualization Technology
6	SMX	Safer Mode Extensions	The processor supports Intel Trusted Execution Technology
7	EIST	Enhanced Intel SpeedStep Technology	The processor supports Enhanced Intel SpeedStep Technology and implements the IA32_PERF_STS and IA32_PERF_CTL registers.
8	TM2	Thermal Monitor 2	The processor implements the Thermal Monitor 2 thermal control circuit (TCC).
9	SSSE3	Supplemental Streaming SIMD Extensions 3	The processor supports the Supplemental Streaming SIMD Extensions 3 instructions.
10	CNXT-ID	L1 Context ID	The L1 data cache mode can be set to either adaptive mode or shared mode by the BIOS.
11		Reserved	-
12	FMA	Fused Multiply Add	The processor supports FMA extensions using YMM state.
13	CX16	CMPXCHG16B	The processor supports the CMPXCHG16B instruction.
14	xTPR	xTPR Update Control	The processor supports the ability to disable sending Task Priority messages. When this feature flag is set, Task Priority messages may be disabled. Bit 23 (Echo TPR disable) in the IA32_MISC_ENABLE MSR controls the sending of Task Priority messages.

## Продолжение таблицы Г.2

Bit	Name	Description (when Flag = 1)	Comments
15	PDCM	Perfmon and Debug Capability	The processor supports the Performance Capabilities MSR. IA32_PERF_CAPABILITIES register is MSR345h.
16		Reserved	-
17	PCID	Process Context Identifiers	The processor supports PCIDs and that software may set CR4.PCIDE to 1.
18	DCA	Direct Cache Access	The processor supports the ability to prefetch data from a memory mapped device.
19	SSE4.1	Streaming SIMD Extensions 4.1	The processor supports the Streaming SIMD Extensions 4.1 instructions.
20	SSE4.2	Streaming SIMD Extensions 4.2	The processor supports the Streaming SIMD Extensions 4.2 instructions.
21	x2APIC	Extended xAPIC Support	The processor supports x2APIC feature.
22	MOVBE	MOVBE Instruction	The processor supports MOVBE instruction.
23	POPCNT	POPCNT Instruction	The processor supports the POPCNT instruction.
24	TSC-DEADLINE	Time Stamp Counter Deadline	The processor's local APIC timer supports one-shot operation using a TSC deadline value.
25	AES	AES Instruction Extensions	The processor supports the AES instruction extensions.
26	XSAVE	XSAVE/XSTOR States	The processor supports the XSAVE/XRSTOR processor extended states feature, the XSETBV/XGETBV instructions, and the XFEATURE_ENABLED_MASK register (XCR0)
27	OSXSAVE	OS-Enabled Extended State Management	The OS has enabled XSETBV/XGETBV instructions to access the XFEATURE_ENABLED_MASK register (XCR0), and support for processor extended state management using XSAVE/XRSTOR.
28	AVX	Advanced Vector Extensions	The processor supports the AVX instruction extensions.
29	F16C	16-bit floating-point conversion instructions	The processor supports 16-bit floating-point conversion instructions.
30	RDRAND	RDRAND instruction supported	The processor supports RDRAND instruction.
31		Not Used	Always returns 0.

## ПРИЛОЖЕНИЕ Д

### ПОРЯДОК РАЗРАБОТКИ АССЕМБЛЕРНОЙ DOS-ПРОГРАММЫ

Разработка и выполнение DOS-программ проводится на ПЭВМ IBM PC AT в среде MS-DOS. В качестве инструментальных программных средств используются обрабатывающие программы пакета Turbo Assembler фирмы Borland: компилятор `tasm.exe`, компоновщик `tlink.exe`. Для отладки программ необходимо воспользоваться стандартным отладчиком `td.exe` фирмы Borland.

Порядок разработки ассемблерной программы следующий.

Шаг 1. Используя текстовый редактор, который позволяет сохранять информацию в виде текста (коды ASCII), набрать эту программу и сохранить на локальном жестком диске в рабочем каталоге системы Турбо Ассемблер (`C:\TASM\BIN\`) под именем **имя\_файла.asm**, где **имя\_файла** включает в себя: обозначение специальности (в 1-3 позиции), год поступления в ХНУРЭ (4-5 поз.), номер группы (6 поз.), номер бригады (7 поз.), номер работы (8 поз.), например: `ki_15181.asm`.

Шаг 2. Выполнить компиляцию программы. С этой целью используется вызов компилятора из командной строки:

```
tasm /l имя_файла[.asm],
```

где /l – параметр вызова, задающий формирование листинга трансляции (файл **имя\_файла.lst**).

В результате будет сформирован файл объектного кода программы (**имя\_файла.obj**) и файл листинга трансляции. При наличии ошибки компилятор отобразит строку, в которой она обнаружена, а файл объектного кода не будет сгенерирован. В этом случае необходимо скорректировать исходный текст и повторить компиляцию. Если же ошибок нет, следует перейти к шагу 3.

Шаг 3. Выполнить компоновку программы. С этой целью используется вызов компоновщика из командной строки:

```
tlink имя_файла[.obj],
```

с созданием файла исполнимого кода (**имя\_файла.exe**).

Если ошибок нет – перейти к шагу 4, в противном случае – вернуться к шагу 1.

Шаг 4. Выполнить отладку исполнимого файла:

```
td имя_файла[.exe].
```

С целью автоматизации процесса создания исполнимого файла, рекомендуется разработать пакетный файл следующего содержания:

```
tasm /l имя_файла  
tlink имя_файла
```

Пакетный файл должен иметь имя, отличное от имени ассемблерной программы, и расширение .bat.