Харьковский национальный университет радиоэлектроники
Кафедра ЭВМ
ТОРБА
Александр Алексеевич
Кандидат технических наук, профессор

Украина, 61166, г.Харьков, пр. Ленина 14, ауд. 221

Раб.тел: (8-057) 702-13-54
Email: august.al@yandex.ua

КОНСПЕКТ ЛЕКЦИЙ
И
МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к лабораторной работе
по программированию
ПРОЦЕССОРА ЧИСЕЛ С ПЛАВАЮЩЕЙ ТОЧКОЙ x87 (F P U)

Утверждено на заседании каф. ЭВМ
Протокол №__ от «___» _____ 2017

Зав. кафедрой ЭВМ,

проф. Коваленко А.А.

2021

3.1 ПРОЦЕССОР ЧИСЕЛ С ПЛАВАЮЩЕЙ ТОЧКОЙ (FPU)

3.1.1 ФОРМАТЫ ЧИСЕЛ С ПЛАВАЮЩЕЙ ТОЧКОЙ

Сопроцессор x87 распознает три формата чисел с плавающей точкой, хранимых в памяти (рис. 3.1). Внутри сопроцессора все числа преобразуются в расширенный формат.

Каждый формат состоит из трех полей: знак (S), порядок и мантисса (рис. 3.1). 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.XXXXXXX...XXX , где: X = 0 или 1

Рис. 3.1 – *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: $(-1)^S (1 . X1\ X2 \ldots X23) * 2^{(E - 127)}$,

Двойная точность: $(-1)^S (1 . X1\ X2 \ldots X52) * 2^{(E - 1023)}$,

Расширенная точность: $(-1)^S (X1 . X2 \ldots X64) * 2^{(E - 16383)}$,

где S – значение знакового бита;
X1 X2 .. – 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.

Таблица 3.1 – *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

Диапазон представления чисел с плавающей точкой в десятичной системе счисления приведен в табл. 3.1 и рис. 3.2.

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 АНТИПЕРЕ-ПОЛНЕНИЕ.

Отметим, что рис. 3.2 абсолютно симметричен для положительных и отрицательных чисел. Следовательно, операция нахождения абсолютного значения никогда не может вызвать ни переполнения, ни антипереполнения.

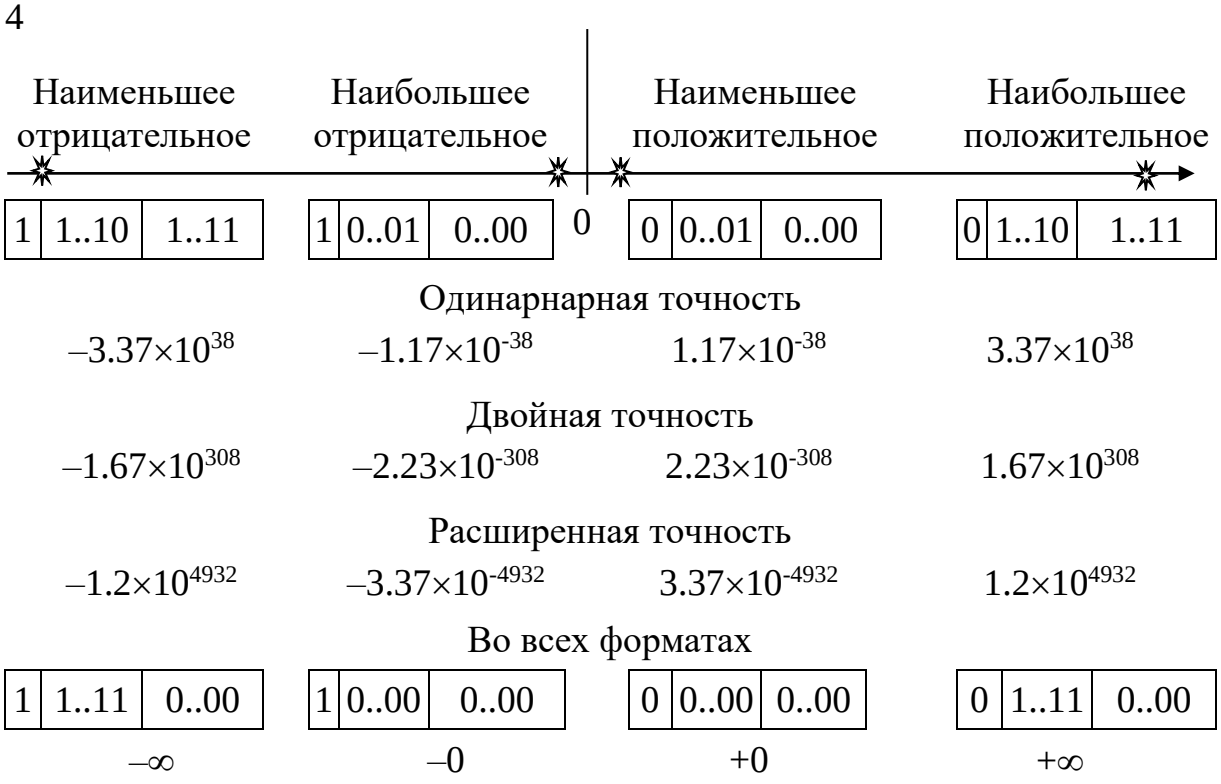

Рис. 3.2 – *Диапазон представления чисел*

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. 3.3:

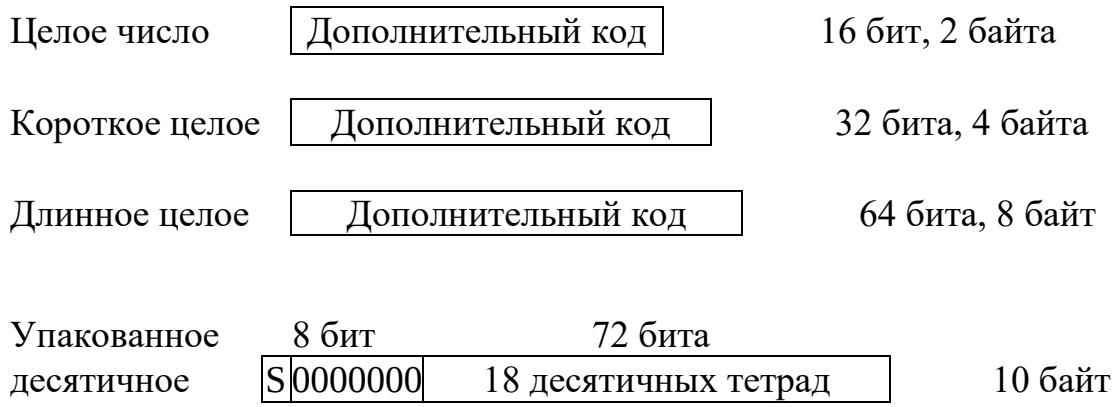

Рис 3.3 – *Форматы целых чисел*

ЦЕЛОЕ СЛОВО – это 16-ти 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:

- 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. 3.16) очевидным образом приводит к знаковым бесконечностям:

$$x / (+0) = +\infty, \qquad -x / (+0) = -\infty,$$
$$x / (-0) = -\infty, \qquad -x / (-0) = +\infty.$$

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. 3.2) и два нуля; здесь все конечные числа «*x*» 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.
- ЧИСЛЕННОЕ 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.

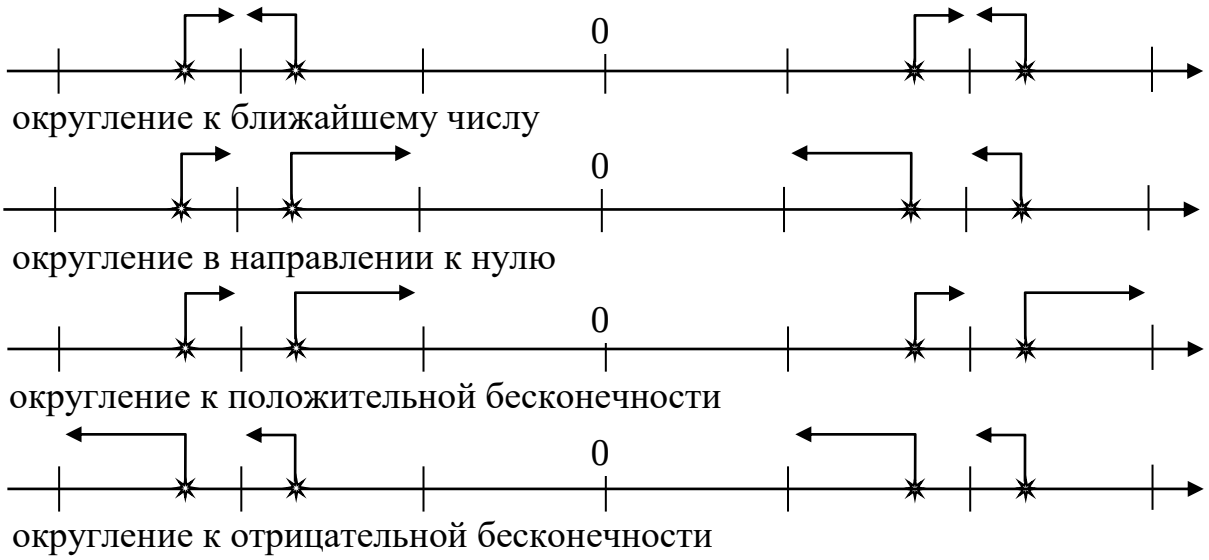

Рис. 3.4 – *Режимы округления неточного результата*

3.1.2 РЕГИСТРЫ СОПРОЦЕССОРА x87

Сопроцессор x87 имеет регистры (рис. 3.5), 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. 3.6).

16-ти битовый **РЕГИСТР УПРАВЛЕНИЯ** содержит поле МАСОК ОСОБЫХ СЛУЧАЕВ и поля округления чисел (рис. 3.7).

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).

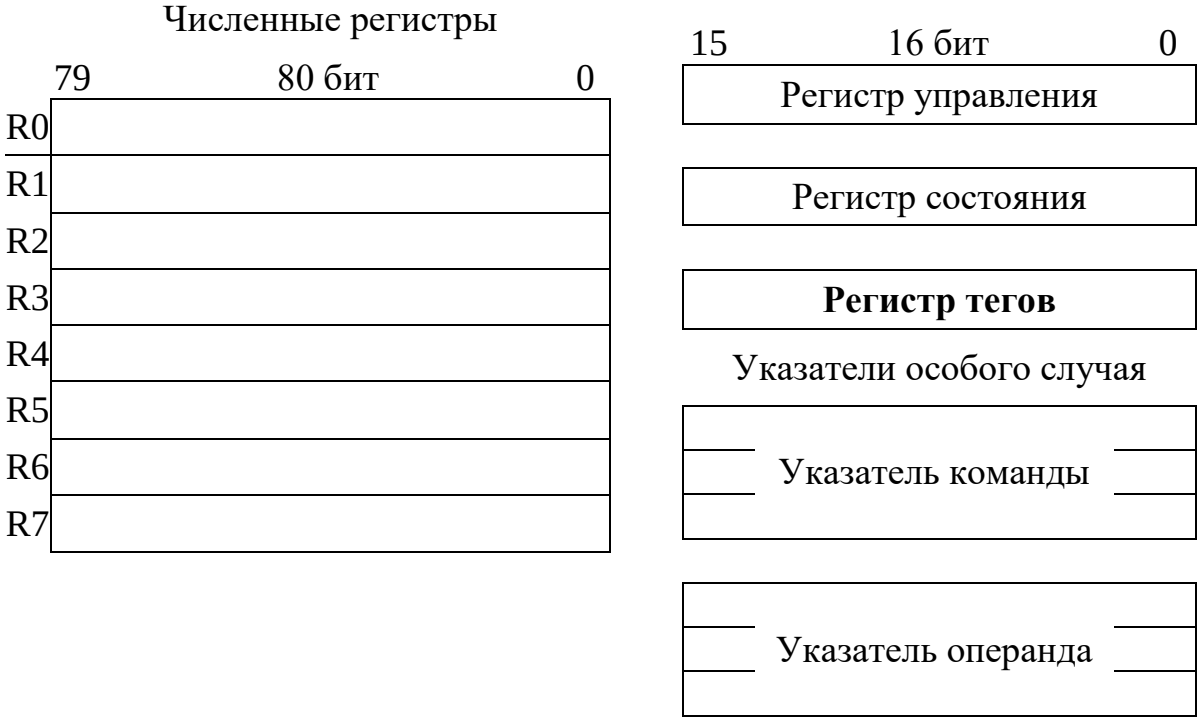

Рис. 3.5 – *Регистры сопроцессора x87*

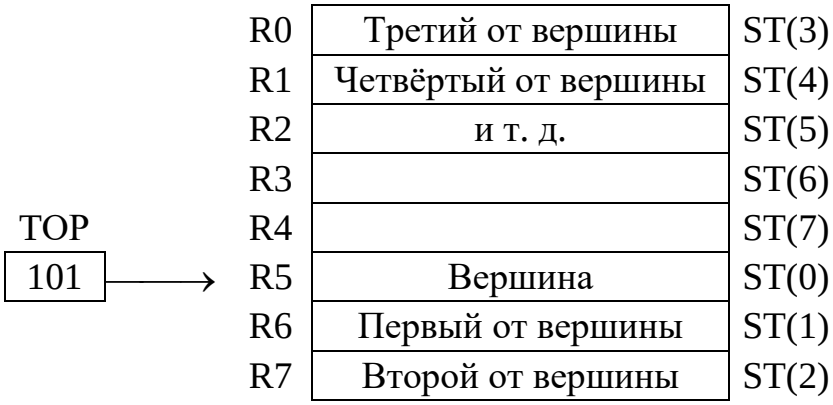

Рис. 3.6 – *Адресация численных регистров как стека*

16-ти битовый **РЕГИСТР СОСТОЯНИЯ** содержит флаги, модифицируемые после выполнения команд, а также поле вершины стека (TOP) (см. рис. 3.7).

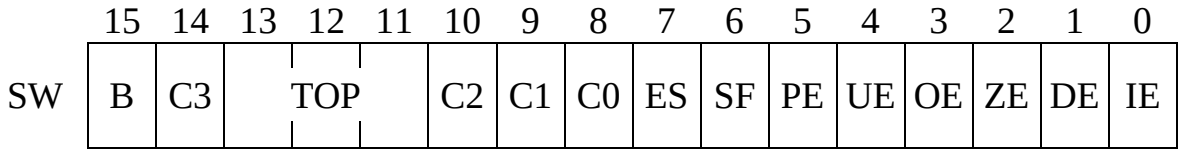

Рис. 3.7 – *Регистр состояния (Status Word)*

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. 3.6). 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.

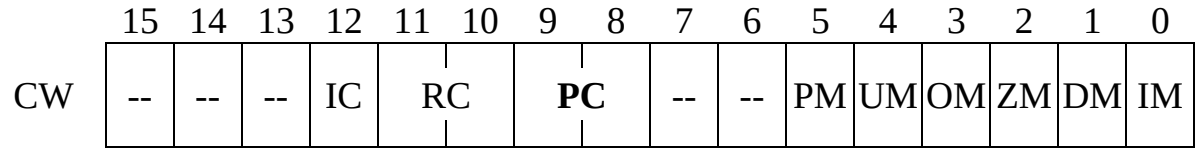

Рис. 3.8 – *Регистр управления (Control Word)*

16-ти 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:

- PC = 11 – округление до расширенной точности (принимается по умолчанию),
- PC = 10 – округление до двойной точности,
- PC = 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. 3.4):

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

Бит УПРАВЛЕНИЯ БЕСКОНЕЧНОСТЬЮ – IC – задает режим интерпретации бесконечности:

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

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

16-ти битовый **РЕГИСТР ТЭГОВ** состоит из 8-ми двухбитовых полей (рис. 3.9). 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.

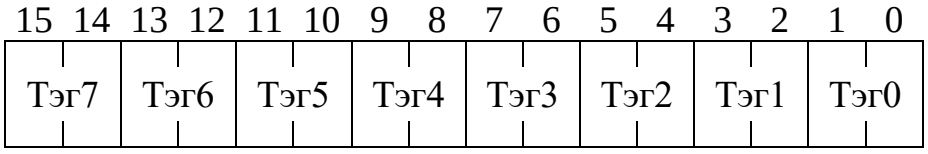

Рис. 3.9 – *Регистр тэгов*

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, сопроцес-

сор 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. 3.10).

Рис. 3.10 – *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-ти битовые адреса этих объектов, а также 11 младших бит кода операции.

3.1.3 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** (Poping) 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.

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 3.2 – *Команды передачи данных*

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

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 – reveres):

ПРИЕМНИК ← ИСТОЧНИК **$** ПРИЕМНИК.

ВО ВСЕХ КОМАНДАХ ОДИН ОПЕРАНД ДОЛЖЕН БЫТЬ В ВЕРШИНЕ СТЕКА. Мнемоники всех базовых арифметических команд приведены в табл. 3.3.

Таблица 3.3 – *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. 3.11).

- 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. 3.12). Последняя форма является частным случаем предыдущей.

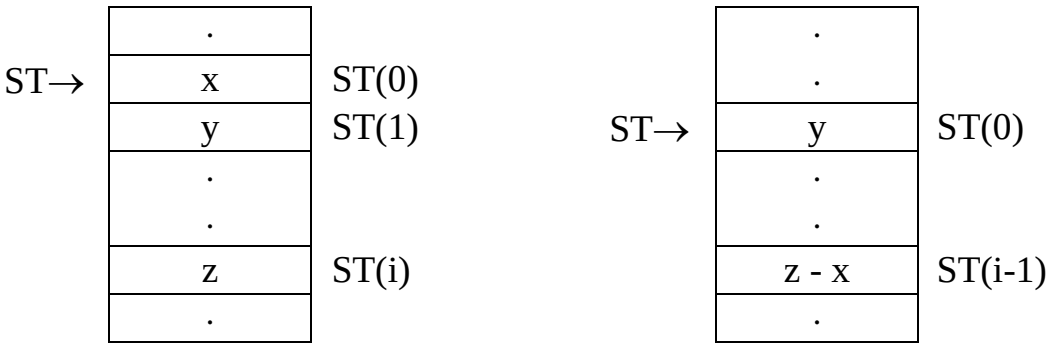

Рис. 3.11 – *Действие команды FSUBP ST(i),ST*

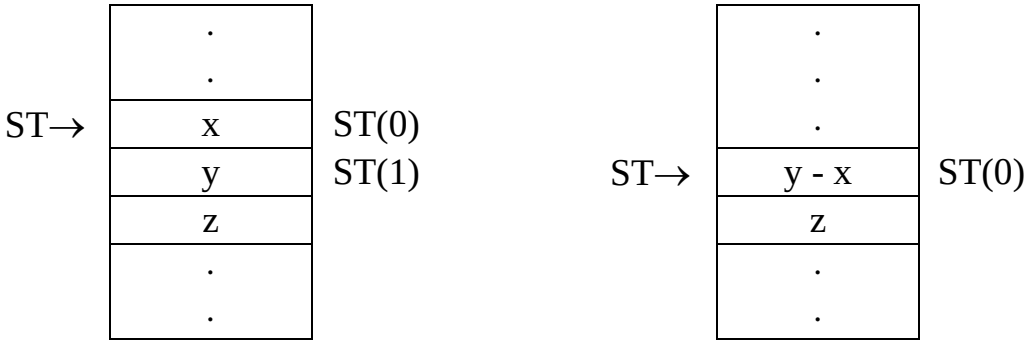

Рис. 3.12 – *Действие команды FSUB*

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.

КОМАНДЫ СРАВНЕНИЙ

- 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 3.4).

Таблица 3.4 – *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

- 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+).
- 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.

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 3.5 – *Трансцендентные команды*

Мнемоника	Описание команды	Вычисляемая функция
FPTAN	Частичный тангенс	ST(1) / ST(0) = 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) = 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):

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

Число «*y*» заменяет старое содержимое ST(0), а число «*x*» включается сверху. Поэтому, после выполнения команды указатель стека умень-

шится на 1, число «*x*» будет записано в новую вершину стека ST(0), а число «*y*» – 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 находится в диапазоне (сопроцессоры 87/287):

$$0 <= ST(0) <= 1/4.$$

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

$$-2^{63} <= ST(0) <= +2^{64}.$$

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. 3.6). 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: $0 \le z \le 1/4$.

Таблица 3.6 – *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: $-2^{63} <= ST(0) <= +2^{63}$. 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. 3.7 (***делить операнды не нужно***).

Таблица 3.7 – *Формулы для вычисления обратных тригонометрических функций*

$arcSin(z) = arctg\left(\frac{z}{\sqrt{(1-z^2)}}\right)$ ←ST(1) ←ST(0)	
$arcCos(z) = 2 * arctg\left(\frac{\sqrt{(1-z)}}{\sqrt{(1+z)}}\right)$ ←ST(1) ←ST(0)	
$arctg(z) = arctg\left(\frac{z}{1}\right)$ ←ST(1) ←ST(0)	$arcCtg(z) = arctg\left(\frac{1}{z}\right)$ ←ST(1) ←ST(0)
$arcCo\sec(z) = arctg\left(\frac{sign(z)}{\sqrt{(z^2-1)}}\right)$ ←ST(1) ←ST(0)	
$arc\sec(z) = 2 * arctg\left(\frac{\sqrt{(\lvert z\rvert-1)}}{\sqrt{(\lvert z\rvert+1)}}\right)$ ←ST(1) ←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. 3.8 с загрузкой в регистр ST(1) необходимых констант командами: FLDLN2 и FLDLG2.

Таблица 3.8 – *Формулы для вычисления логарифмических функций*

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

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 находится в диапазоне :

для сопроцессоров 87/287: $0 <= ST(0) <= 0.5$;

для сопроцессоров 387+: $-1 <= ST(0) <= +1$.

Вычисление функции $2^x - 1$ вместо функции 2^x позволяет избежать потери точности, когда аргумент «x» близок к 0 (а значение функции 2^x близко к 1). Остальные показательные функции вычисляются по формулам в табл. 3.9.

Таблица 3.9 – *Формулы для вычисления показательных функций*

$2^x = [2^x - 1] + 1 = F2XM1 (x) + 1 ;$
$e^x = 2^{(x * log2(e))} = F2XM1 (x * log_2(e)) + 1;$
$10^x = 2^{(x * log2(10))} = F2XM1 (x * log_2(10)) + 1;$
$a^x = 2^{(x * log2(a))} = F2XM1 (x * log_2(a)) + 1 .$

Константы $log_2(e)$ и $log_2(10)$ уже имеются в сопроцессоре и загружаются соответствующими командами (см. таблицу 3.2)

Таблица 3.10 – *Формулы для вычисления гиперболических функций*

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

Таблица 3.11 – *Формулы для вычисления обратных гиперболических функций*

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

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. 3.12. Такое же действие производит аппаратный сигнал сброса – RESET.

Таблица 3.12 – *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"

3.2 ПРАКТИЧЕСКИЕ ЗАДАНИЯ ПО ПРОГРАММИРОВАНИЮ FPU

Задание № 1. Вычислить значение функции :

$$S = \frac{1}{2} * A * B * \sin(\alpha)$$

```
void main ()                // начало программы на языке C++
{
long   A=20, B=30, M2=2;    // описание операндов в памяти
float  ALPHA=0.7, Y;

__asm{                      ; начало ассемблерной вставки
finit                       ; очистка регистров сопроцессора
fld    ALPHA                ; загрузка в вершину стека аргумента
fsin                        ; вычисление синуса
fimul  A                    ; умножение вершины стека на константу
fimul  B
fidiv  M2
fstp   Y                    ; сохранение результата в ячейке памяти
}                           // окончание ассемблерной вставки
}                           // окончание программы на языке C++
```

Задание № 2. Вычислить значение функции:

$$S = \sqrt{p * (p - A) * (p - B) * (p - C)}$$

```
void main ()                // начало программы на языке C++
{
    long   A=20, B=30, C=40, M2=2;   // описание операндов в памяти
    float  RES;             // ячейка памяти для результата

    __asm{                  ; начало ассемблерной вставки
    finit                   ; очистка регистров сопроцессора
    fild       A            ;
    fild       B            ;
    fild       C            ;
    fld        ST           ;          ST(0) →  | p       |
    fadd       ST,ST(2)     ;          ST(1)    | A  (p–A) |
    fadd       ST,ST(3)     ;          ST(2)    | B  (p–B) |
    fidiv      M2           ;          ST(3)    | C  (p–C) |
    fsubr      ST(1),ST     ;
    fsubr      ST(2),ST     ;
    fsubr      ST(3),ST     ;
    fmul                    ; аналог команды   fmulp   ST(1),ST(0)
```

```
        fmul
        fmul
        fsqrt
        fstp    RES             ; сохранение результата и очистка ST(0)
        }                       // окончание ассемблерной вставки
}                               // окончание программы на языке C++
```

Задание №3. Определить номер (n) элемента прогрессии : $a_n = 5{,}3^n + 5 * n$, при котором сумма элементов превысит 20000.

```
void main ()                    // начало программы на языке C++
{
        float   A=5.3;          // описание операндов в памяти
        long    B=5, C=20000, N=0;

        __asm{                  ; начало ассемблерной вставки
        finit                   ; очистка регистров сопроцессора
        fld1                    ; регистр для вычисления степенной функц.
        fldz                    ; регистр для накопления суммы прогрессии
m1:     inc     N               ;наращивание аргумента
        fld     A               ; загрузка в ST(0) 5,3
        fmulp   ST(2),ST        ; вычисление степенной функции  5,3^n
        fild    B
        fimul   N
        fadd    ST,ST(2)        ; вычисление элемента прогрессии
        fadd                    ; накопление суммы прогрессии
        ficom   C               ; сравнение суммы с 20000
        fstsw   AX              ; сохранение регистра состояния  SW (FPU) в
                                ; регистре  AX (CPU)
        sahf                    ;сохранение старшего байта AX в рег. флагов
        jc      m1              ; переход, если сумма меньше 20000
        }                       // окончание ассемблерной вставки
}                               // окончание программы на языке C++
```

Задание № 4. Вычислить функцию Y. Результат перевести в градусы. $Y = 5 * arcSin ((tg\ 60°)^2 / 7)$.

```
void main ()                    // начало программы на языке C++
{
        long    A=60, B=5, C=7, D=180;      // описание операндов в памяти
        float   Y;
```

```
    __asm{                  ; начало ассемблерной вставки
    finit                   ; очистка регистров сопроцессора
    fldpi                   ; загрузка в вершину стека числа 3,1415…
    fimul       A           ; умножение числа «пи» на аргумент
    fidiv       D           ; деление аргумента на 180
    fptan                   ; вычисление частичного тангенса
    fdiv                    ; нахождение тангенса
    fmul        ST,ST       ; возведение в квадрат
    fidiv       C           ; вычислен «z» – аргумент ArcSin
    fld         ST          ; копирование вершины стека
    fmul        ST,ST       ; возведение в квадрат
    fld1                    ; включение в стек «1»
    fsubr                   ; вычитание с реверсом : 1 – z^2
    fsqrt                   ; корень квадратный
    fpatan                  ; вычисление арктангенса
    fimul       B           ; умножение на константу
    fldpi                   ;
    fdiv                    ; } перевод из радианов – в градусы
    fimul       D           ;
    fstp        Y           ; сохранение результата в ячейке памяти
    }                       // окончание ассемблерной вставки
}                           // окончание программы на языке C++
```

Задание № 5. Возведение основания 2 в произвольную степень. Фрагмент программы :

В вершине стека находится аргумент «z».
Результат (2^z) помещается в вершину стека.

```
fld      ST(0)          ;  Копирование вершины стека
frndint                 ;  Округление  ST(0)  до целого
fsub     ST(1),ST(0)    ;  Выделение дробной части в ST(1)
fxch                    ;  Обмен регистров ST(0), S(1)
f2xm1                   ;  Возведение 2 в дробную степень (минус 1)
fld1                    ;  Загрузка константы  «1»
fadd                    ;  Добавление единицы
fscale                  ;  Возведение 2 в целую степень и умножение
fstp     ST(1)          ;  Удаление регистра и сдвиг вершины. Резуль-
                        ;  тат в вершине стека
```

3.3 ЛАБОРАТОРНАЯ РАБОТА N 3

3.3.1 ЦЕЛЬ РАБОТЫ

- углубить и закрепить знания по архитектуре арифметических сопроцессоров 8087+ и навыки по их программированию;
- приобрести практические навыки в составлении, отладке и выполнении программ, написанных на языке ассемблера для арифметических сопроцессоров 8087+.

3.3.2 САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ

Перед выполнением лабораторной работы студентам необходимо изучить программную модель и систему команд языка ассемблера арифметических сопроцессоров x87+ (FPU).

Изучить основные сведения о работе с программной средой Visual C++, функциональные возможности и режимы работы программы-отладчика.

3.3.3 ОТЛАДКА ПРОГРАММ НА ЯЗЫКЕ АССЕМБЛЕР В ПРОГРАММНОЙ СРЕДЕ VISUAL C++

После запуска программы «Visual C++»:

- в меню «File» выбрать команду «New»,
- в открывшемся окне выбрать закладку «Projects»,
- на этой закладке выбрать «Win32 Console Application»,
- в поле «Project name» записать имя проекта (например: lab_2_1),
- в поле «Location» выбрать папку для записи проекта.

После формирования папки проекта необходимо ввести программу на языке «C++» с ассемблерной вставкой. Для этого:

- в меню «File» выбрать команду «New»,
- в открывшемся окне выбрать закладку «Files»,
- на этой закладке выбрать «C++ Source File»,
- в поле «File name» записать имя файла (например: lab_2_1),.

После ввода программы необходимо ее сохранить («CTR-S»).

Для компиляции программы – нажать клавишу «F5».

Для пошаговой отладки программы необходимо:

- установить курсор в начале первой строки ассемблерной вставки,
- в меню «Build» выбрать команду «Start Debug» и вариант «Run to Cursor» (или нажать «CTR-F10»).

Каждый шаг отлаживается нажатием на кнопку «F10».

3.3.4 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

3.3.4.1 Исследовать выполнение арифметических операций. Номер варианта выбирается в соответствии с последней цифрой номера зачетной книжки.

Вариант 1. Вычислить 6 значений функции:

$Y = 3,5 * x^2 + 7,2 * x$ (x – изменяется от 4,5 с шагом 3). Результат округлить до целого и разместить в памяти.

Вариант 2. Вычислить 5 значений суммы прогрессии :

$a_n = 2,5 * n^2 + 5,3$ (для n – от 4 с шагом 1). Результат разместить в памяти в формате INTEGER.

Вариант 3. Вычислить 7 значений функции:

$Y = 2500 / (2 * x^2 + 3,7)$ (x – изменяется от 3 с шагом 2,5). Результат округлить до целого и разместить в памяти.

Вариант 4. Вычислить 6 значений функции:

$Y = \sqrt{14 * x^2 + 5,6}$ (x – изменяется от 6 с шагом 3). Результат разместить в памяти.

Вариант 5. Вычислить 6 значений функции :

$Y = (20 * x) /(5 * x^2 – 8,5)$ (x – изменяется от 1 с шагом 4). Результат разместить в памяти.

Вариант 6. Вычислить 6 значений суммы прогрессии :

$a_n = (4,5^n) / (n + 5)$ (для n – от 1 с шагом 1). Результат округлить до целого и разместить в памяти.

Вариант 7. Вычислить 5 значений функции:

$Y = (x – 7) / \sqrt{x^2 + 20}$ (x – изменяется от 3 с шагом 2,5). Результат разместить в памяти.

Вариант 8. Вычислить 6 значений функции :

$Y = 1024 / (3,2 * x^2 – 25),$ (x – изменяется от 2 с шагом 2). Результат разместить в памяти.

Вариант 9. Вычислить 5 значений суммы прогрессии :

$a_n = 418 / (2 * n^2 + 7,3),$ (для n – от 1 с шагом 1). Результат разместить в памяти.

Вариант 10. Вычислить 6 значений функции :

$Y = 3000 / (x^2 + 3,6 * x – 7,5),$ (x – изменяется от 2 с шагом 2,7). Результат разместить в памяти.

3.3.4.2 Исследовать выполнение операций сравнения.

Вариант 1. Найти целое значение аргумента, при котором функция $Y = 20 / (x^2 + 2{,}5^x)$ станет меньше 0,2.

Вариант 2. Найти целое значение аргумента x, при котором функция $Y = 15 / (x^2 + 3{,}7)$ станет меньше 0,1 (для x – от 1 с шагом 1).

Вариант 3. Определить номер (n) элемента прогрессии :

$a_n = 2{,}5 * n^2 + 7{,}3$, при котором сумма элементов прогрессии превысит 1000.

Вариант 4. Определить номер (n) элемента прогрессии :

$a_n = 3{,}3^n + 5$, при котором сумма элементов превысит 15000.

Вариант 5. Задан массив с элементами : $a(i) = \sin (5 * i)$.

Определить номер элемента массива, при котором сумма элементов превысит 3. Аргумент синуса задан в градусах.

Вариант 6. Найти целое значение аргумента, при котором функция

$Y = \sqrt{2 * 3{,}5^x + 10}$ превысит 100.

Вариант 7. Определить номер (n) элемента прогрессии :

$a_n = \sqrt{2{,}5^n + 3 * n}$, при котором сумма элементов превысит 100.

Вариант 8. Задан массив с элементами : $b(i) = \sin(2 * i^2)$.

Определить номер элемента, при котором сумма элементов превысит 3. Аргумент синуса задан в градусах.

Вариант 9. Найти целое значение аргумента, при котором функция

$Y = (5{,}6^x) / (3 * x^2)$ превысит 200.

Вариант 10. Найти целое значение аргумента x, при котором функция $Y = \sqrt{15 * x^2 + 32 * x + 40}$ превысит 30.

3.3.4.3 Исследовать выполнение команд с обратными тригонометрическими функциями. Вычислить ***одно значение*** функции Y. Параметры аргументов выбирать с учетом области определения. ***Результат перевести в градусы.***

Вариант 1. $Y = 3 * \arcsin (2 * \cos(70°))^2$.

Вариант 2. $Y = (1/2) * \operatorname{arccosec} (3 * A^2 + 4 * B)$.

Вариант 3. $Y = 2 * \arcsin ((A^2 + B^2)/3)$.

Вариант 4. $Y = 3 * arcCos\ (2 * A^2 - B)$.

Вариант 5. $Y = 5 * arcSec\ (3 * (tg\ 70°)^2)$.

Вариант 6. $Y = (1/3)\ arcSin\ (3 * A + sin\ 20°)$.

Вариант 7. $Y = 4 * arcCos\ (2 * sin\ 30° * cos\ 30°)$.

Вариант 8. $Y = (1/4)\ arcCosec\ ((5 * A + B^2)/2)$.

Вариант 9. $Y = 3 * arcCosec\ (4 * A + tg\ 40°)$

Вариант 10. $Y = 5 * arcSin\ (3 * cos\ 25° * sin\ 25°)$.

3.3.4.4 Исследовать выполнение команд с логарифмическими и показательными функциями.

Вариант 1. Вычислить 6 значений функции: $Y = 5 * ln\ (sin\ x)$, x – изменяется в градусах от 10 с шагом 15.

Вариант 2. Вычислить 5 значений функции: $Y = 7^x$, x – изменяется от 0,5 с шагом 0,2.

Вариант 3. Вычислить 7 значений функции : $Y = 4 * lg\ (tg\ x)$, x – изменяется в градусах от 15 с шагом 10.

Вариант 4. Вычислить 6 значений функции: $Y = 12^x$, x – изменяется от 0,5 с шагом 0,3.

Вариант 5. Вычислить 5 значений функции: $Y = 3 * log_8(x^2 + 1)$, x – изменяется от 0,2 с шагом 0,3.

Вариант 6. Вычислить 7 значений функции: $Y = 5^{(sin\ x)}$, x – изменяется в градусах от 10 с шагом 8.

Вариант 7. Вычислить 6 значений функции : $Y = 7 * \ln(x^2 + \sqrt{x})$, x – изменяется от 2 с шагом 3.

Вариант 8. Вычислить 5 значений функции : $Y = 4\wedge(x^2 + 1)$, x – изменяется от 0,2 с шагом 0,4.

Вариант 9. Вычислить 7 значений функции: $Y = 6 * lg\ (cos\ x)$, x – изменяется в градусах от 8 с шагом 12.

Вариант 10. Вычислить 6 значений функции : $Y = 3^{(cos\ x)}$, x – изменяется в градусах от 10 с шагом 8.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Форматы данных сопроцессора.
2. Программная модель сопроцессора. Адресация регистров.
3. Особые случаи сопроцессора и их маскирование.
4. Кодирование мнемоник сопроцессора.
5. Команды передачи данных сопроцессора.
6. Арифметические команды сопроцессора.
7. Тригонометрические команды.
8. Логарифмические и показательные команды.
9. Команды управления сопроцессором.
10. Перевести число 75,125 в формат двойной точности с плавающей точкой.
11. Перевести число с плавающей точкой C077760000000000h в привычный десятичный формат.

РЕКОМЕНДОВАННАЯ ЛИТЕРАТУРА

1. Гук М., Юров В. Процессоры PETNTIUM 4, ATHLON и другие – СПб: Питер, 2001.– 512с.
2. Юров В. Assembler – СПб.: Питер, 2001.– 624с.
3. Григорьев В.Л. Микропроцессор i486. Архитектура и программирование. В 4-х книгах.-М.: Гранал, 1993.