КОНСТРУИРОВАНИЕ ПРОГРАММ

Лекция № 14 Базовый FPU

+375 17 293 8039 (505a-5)

+375 17 320 7402 (ОИПИ НАНБ)

prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by/

Кафедра ЭВМ, 2022

2022.04.27

Оглавление

Арифметические расширения команд процессора

Особенности и проблемы чисел с плавающей запятой в двоичном представлении

1) разные компьютеры — разные способы представления, соответственно, разные результаты при выполнении вычислений. Например, вычисления на CPU и GPU дадут разные результаты;

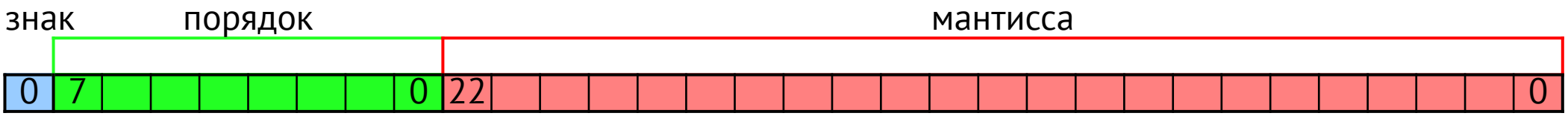
2) неассоциативность — результат вычислений зависит от порядка выполнения действий;

3) ноль имеет знак;

4) разность неравных чисел дает ноль.

Формат числа с плавающей запятой (плавающей точкой) — LK02

Число с плавающей запятой (или число с плавающей точкой) — форма представления действительных чисел, в которой число хранится в форме **мантиссы** и **показателя степени**.



При этом число с плавающей запятой имеет **фиксированную относительную точность** и **изменяющуюся абсолютную**.

Реализация математических операций с числами с плавающей запятой в вычислительных системах может быть как аппаратная, так и программная.

Нормальная форма и нормализованная форма

Нормальная форма числа с плавающей запятой называется такая форма, в которой мантисса (без учёта знака) находится на полуинтервале [0; 1) ($0 \leq a < 1$).

Нормальная форма записи имеет недостаток: некоторые числа записываются неоднозначно (например, 0,0001 можно записать в 4-х формах: $0,0001 \cdot 10^0; 0,001 \cdot 10^1; 0,01 \cdot 10^2; 0,1 \cdot 10^3$.

Нормализованная форма

мантисса десятичного числа принимает значения в диапазоне [1; 10) ($1 \leqslant a < 10$)

мантисса двоичного числа принимает значения в диапазоне [1; 2) ($1 \leqslant a < 2$).

В нормализованной форме любое число кроме 0 записывается единственным образом.

Недостаток: в таком виде невозможно представить 0, поэтому компьютерное представление чисел предусматривает специальный формат представления нуля.

Стандарт IEEE 754

IEEE 754 — технический стандарт формата представления чисел с плавающей точкой, используемый как в программных реализациях арифметических действий, так и во многих аппаратных (CPU и FPU) реализациях. Стандарт описывает:

Определенные числа (finite number) по основанию 2 и 10. Каждое определенное число (как представлено в стандарте) описывается тремя целыми:

s — знак (0 или 1), p — точность (precision), c — мантисса (significand или коэффициент), имеющая не более p цифр по основанию b (т.е. представляющая целое от 0 до $b^p - 1$), и q — показатель (exponent), лежащий в диапазоне $E_{min} \leqslant q + p - 1 \leqslant E_{max}$.

Определенное число представляется как:

$$(-1)^s \times c \times b^{q-p+1},$$

где b — основание (*base* или *radix*), равное 2 или 10. Эффективное значение порядка сдвинуто и равно $q—127$ для чисел одинарной точности (`float`).

Кроме того, существует два нулевых значения, которые называются *нулями со знаком*. Знаковый бит s определяет является ли ноль +0 (положительный ноль) или -0 (отрицательный ноль).

Две бесконечности : $+\infty$ и $-\infty$. +INF и -INF

Денормализованные числа

Неопределенность:

Два типа NaN:

qNaN — тихий, мягкий (quiet) — попав в операцию, возвратит NaN;

sNaN — сигнализирующий (signaling) — попав в операцию, вызовет исключение;

Также стандарт описывает:

Методы, которые используются для преобразования числа в процессе математических операций;

Обработку исключительных ситуаций, таких как деление на нуль, переполнение, потеря значимости, работу с денормализованными числами и т.д.

Операции — арифметические и другие операции по арифметике форматов.

Name	Обычное название	Base	Digits	E min	E max	Decimal digits	Decimal E max
binary16	половинная точность	2	11	−14	+15	3.31	4.51
binary32	одинарная точность	2	24	−126	+127	7.22	38.23
binary64	двойная точность	2	53	−1022	+1023	15.95	307.95
binary128	четырехкрат. точность	2	113	−16382	+16383	34.02	4931.77
binary256	восмикратная точность	2	237	−262142	+262143	71.34	78913.2
decimal32		10	7	−95	+96	7	96
decimal64		10	16	−383	+384	16	384
decimal128		10	34	−6143	+6144	34	6144

Десятичное E max - это $emax \times \log_{10} основание$ —(максимальная степень в десятичном формате)

Машинная эпсилон

В отличие от чисел с фиксированной запятой, сетка чисел, которые способна отобразить арифметика с плавающей запятой, неравномерна — она более густая для чисел с малыми порядками и более редкая — для чисел с большими порядками (Логарифмическая равномерность).

Относительная погрешность записи чисел одинакова и для малых чисел, и для больших. Поэтому можно ввести понятие машинной эпсилон.

Машинной эпсилон называется наименьшее положительное число ε такое, что $1 \oplus \varepsilon \neq 1$ (знаком $\oplus$ обозначено машинное сложение).

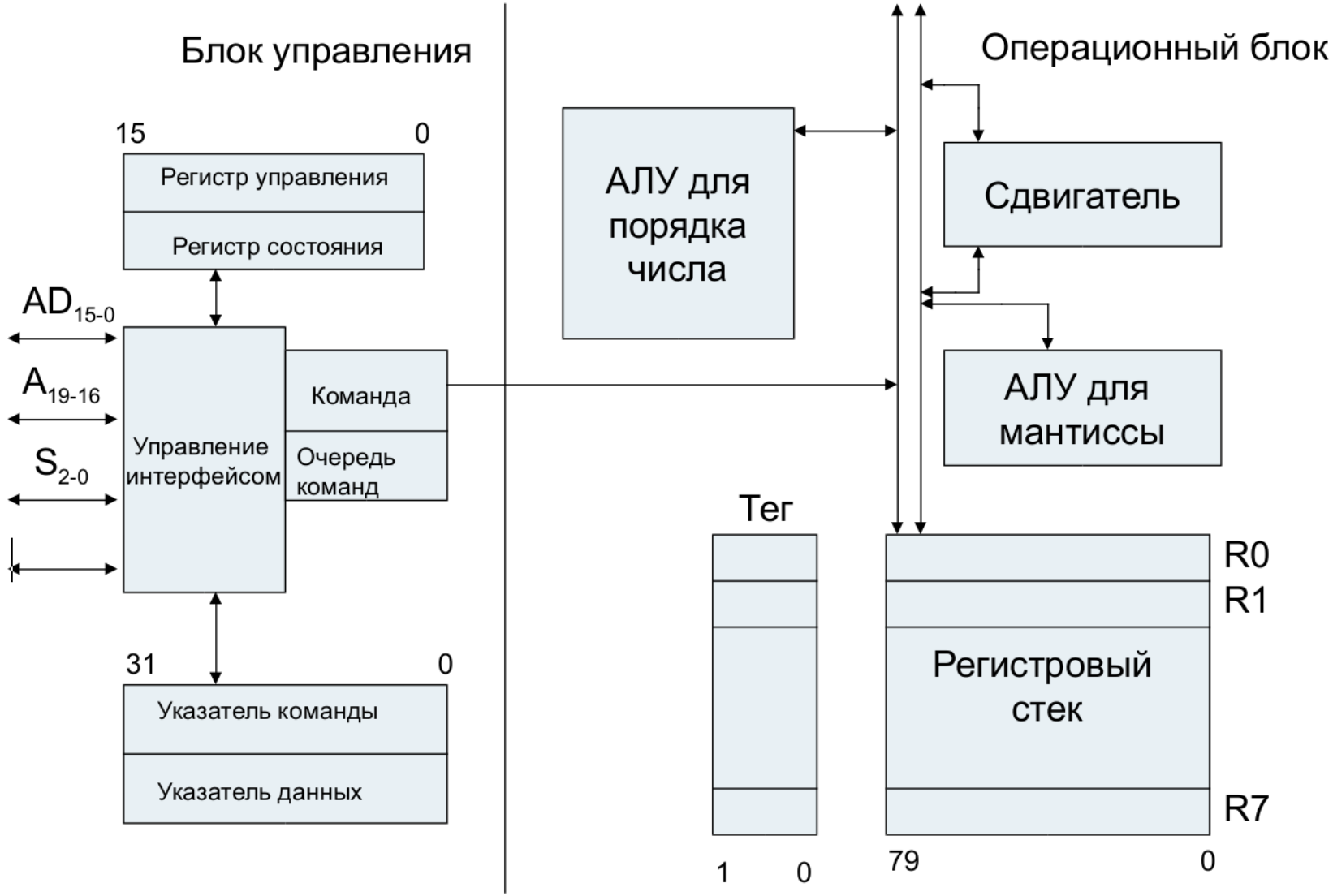
Числа a и b, соотносящиеся так, что $1 < \frac{a}{b} < 1 + \varepsilon$, машина не различает.

В стандарте C/C++ ISO

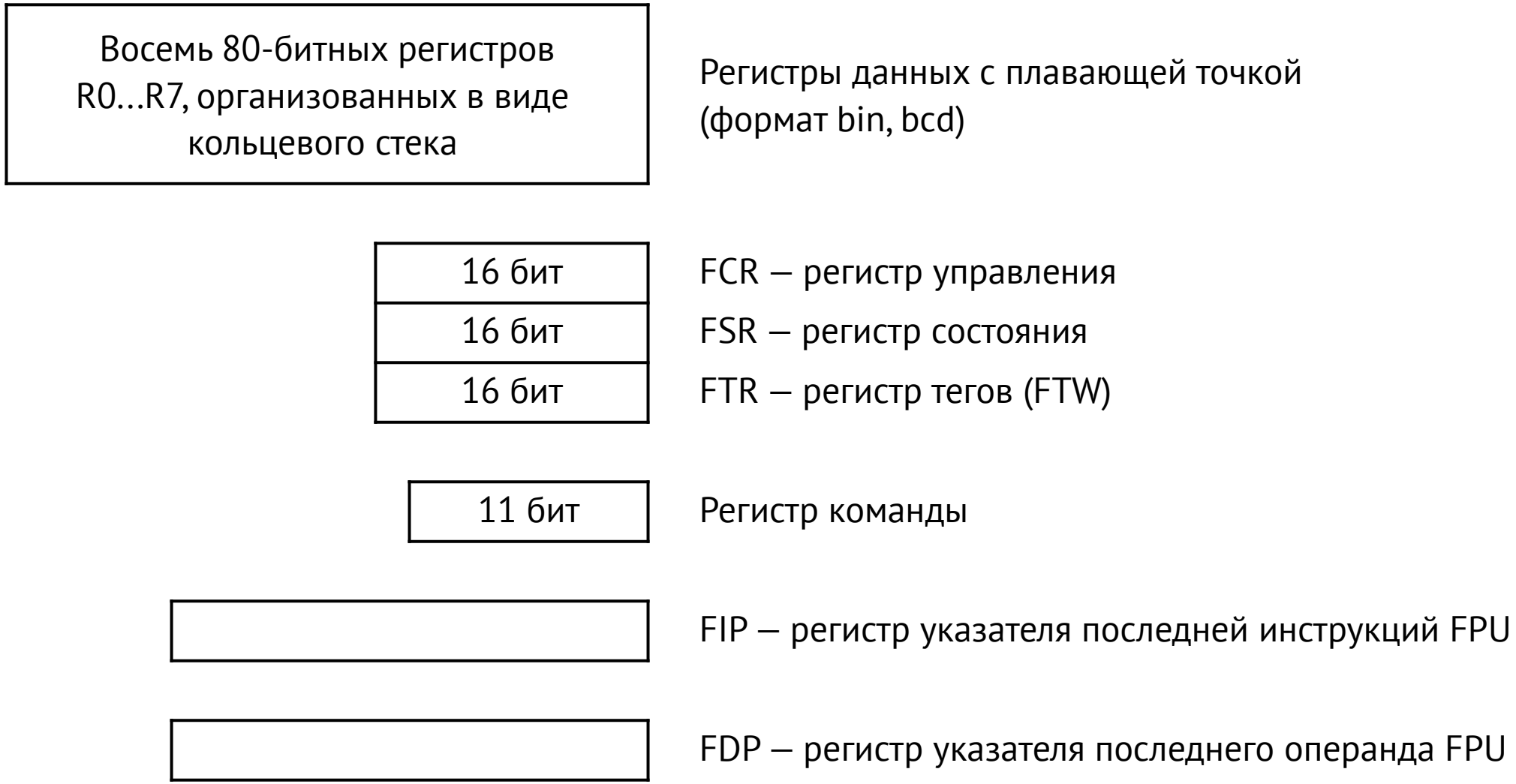
```
#include <cmath> (math.h)
#include <limits> // определяет темплейт класса

std::numeric_limits<double>::epsilon() // ≈ 1 · 10^-16 (1 · 10^-16 ≈ 1 · 2^-53)

#include <cfloat> (float.h)
FLT_EPSILON, DBL_EPSILON, LDBL_EPSILON
```

Сопроцессор арифметики с плавающей точкой (Базовый FPU)

Структура FPU



Регистры FPU



Регистры данных (R0...R7)

Регистры данных не адресуются по именам, а рассматриваются в качестве регистрового стека, вершина которого называется **ST(0)**, соответственно, более глубокие элементы — **ST(1)...ST(7)**.

Регистры организованы в виде кольца.

Если процессор поддерживает **MMX**, мантиссы из регистров стека доступны как **MM0**..**MM7**.

Если выполняется операция загрузки когда **TOP** находится в 0, происходит циклическое сканирование регистров, и новое значение **TOP** устанавливается на 7.

Когда «оборот» может привести к перезаписи несохраненного значения, возникает исключение переполнения стека с плавающей запятой.

Многие инструкции с плавающей запятой имеют несколько режимов адресации, которые позволяют программисту неявно работать с вершиной стека или явно работать с определенными регистрами относительно **TOP**.

Ассемблеры поддерживают эти режимы адресации регистров, используя выражение **ST(0)** или просто **ST** для представления текущей вершины стека и **ST(i)** для указания i-го регистра из **TOP** в стеке ($0 \leqslant i \leqslant 7$). Например, если **TOP** содержит **011B** (регистр 3 — вершина стека), следующая инструкция сложит содержимое двух регистров (3 и 5) в стеке:

```
FADD ST, ST(2)
```

На рисунке ниже показан пример того, как обычно используется структура стека регистров и инструкций FPUx87 для выполнения серии вычислений.

Двумерное скалярное произведение (5.6 x 2.4) + (3.8 x 10.3) вычисляется следующим образом:

```
fld  value1 ; (а) value1 = 5.6
fmul value2 ; (б) value2 = 2.4
fld  value3 ;     value3 = 3.8
fmul value4 ; (в) value4 = 10.3
fadd ST(1)  ; (г)
```

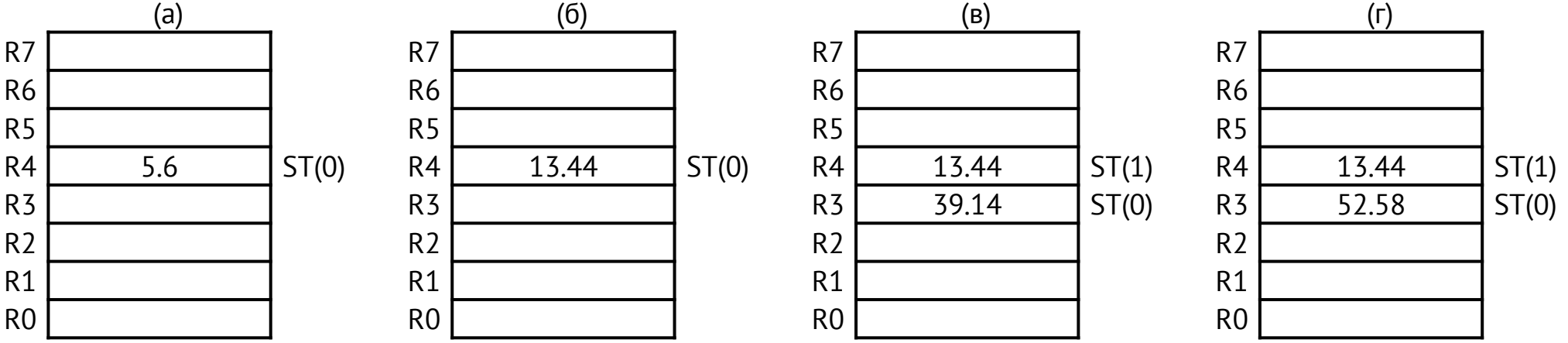
Первая инструкция (**FLD value1**) уменьшает указатель регистра стека (**TOP**) и загружает значение **5.6** в **ST(0)** из памяти (а);

Вторая инструкция умножает значение в **ST(0)** на **2.4** из памяти и сохраняет результат в **ST(0)** (б);

Третья инструкция уменьшает **TOP** и загружает в **ST(0)** значение **3.8**;

Четвертая инструкция умножает значение в **ST(0)** на значение **10.3** из памяти и сохраняет результат в **ST(0)**, (в).

Пятая инструкция складывает значение из **ST(0)** и значение из **ST(1)** и сохраняет результат в **ST(0)**, (г).



Регистр состояний SR содержит слово состояния FPU

бит 15 **B** — занятость FPU. Этот флаг существует для совместимости с i8087, и его значение всегда совпадает с **ES** (бит 7 — флаг ошибки)

бит 14 **C3** — флаг состояния 3;

биты 13..11 TOP — число от 0 до 7, идентифицирующее регистр из **R0...R7**, являющийся вершиной регистрового стека (указывает на **ST(0)**);

бит 10 **C2** — флаг состояния 2;

бит 9 **C1** — флаг состояния 1;

бит 8 **C0** — флаг состояния 0;

Биты **C0...C3** применяются как и биты состояния в основном процессоре — их значения отражают результат выполнения предыдущей команды и используются для условных переходов.

Последовательность команд

```
fstsw ax
sahf
```

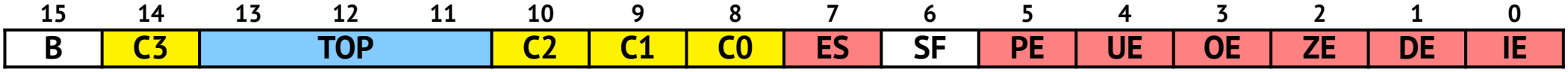
копирует значения битов в регистр **EFLAGS**, при этом:

C0 -> **CF**

C2 -> **PF**

C1 -> теряется

C3 -> **ZF**.



бит 7 ES — общий флаг ошибки.
Если ES = 1, произошло хотя бы одно немаскированное исключение

бит 6 SF — ошибка стека.
C1 = 1, произошло переполнение стека (попытка записи в непустую позицию);
C1 = 0, произошло опустошение (антипереполнение) стека (попытка чтения из пустой позиции).

Биты 0...5 отражают разного рода ошибочные ситуации, которые могут возникать при выполнении команд FPU.

бит 5 **PE** — флаг неточного результата — результат не может быть представлен точно;

бит 4 **UE** — флаг исчезновения порядка (антипереполнения) 0 — результат слишком маленький;

бит 3 **OE** — флаг переполнения 0 — результат слишком большой;

бит 2 **ZE** — флаг деления на 0 — выполенно деление на 0;

бит 1 **DE** — флаг денормализованного операнда — выполнена операция над денормализованным числом;

бит 0 **IE** — флаг недопустимой операции — ошибка стека (SF=1) или выполнена недопустимая операция.

Флаги состояния (Condition Code Flags)

Четыре флага кода состояния (от **C0** до **C3**) содержат результаты сравнений с плавающей запятой и результаты арифметических операций.

Варианты, как инструкции с плавающей запятой устанавливают флаги состояния, приводятся в таблице 8-1 «Condition Code Interpretation» документации Intel на IA-32.

Эти биты кода состояния используются главным образом для условного ветвления и для хранения информации, используемой при обработке исключений.

Флаг кода состояния **C1** используется для множества функций.

Когда в слове состояния FPU установлены флаги **IE** и **SF**, указывающие на переполнение или антипереполнение стека (**#IS**), флаг **C1** позволяет различать переполнение (**C1=1**) и антипереполнение (**C1=0**).

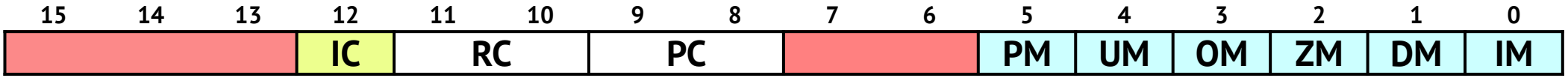
Когда в слове состояния установлен флаг **PE**, что указывает на неточный (округленный) результат, флаг **C1** устанавливается в 1, если последнее округление результата инструкции было направлено вверх.

Инструкция **FXAM** устанавливает **C1** равным знаку тестируемого операнда.

Флаг кода состояния **C2** используется командами **FPREM** и **FPREM1** для указания неполного остатка (частичного остатка). Если в результате получен полный остаток (наименьший вычет), флаг **C2** будет очищен, а флаги **C0**, **C3** и **C1** установлены в три младших значащих бита частного (**Q2**, **Q1** и **Q0** соответственно).

Инструкции **FPTAN**, **FSIN**, **FCOS** и **FSINCOS** устанавливают флаг **C2** в 1, чтобы указать, что операнд источника находится за пределами допустимого диапазона $\pm 2^{63}$, и сбрасывают **C2**, если операнд источника находится в допустимом диапазоне.

Регистр управления CR



биты 15..13 зарезервированы;

бит 12 **IC** — управление бесконечностью. Поддерживается для совместимости с i8087 и i80287 — вне зависимости от значения этого бита $+\infty > -\infty$;

биты 11..10 **RC** — управление округлением (rounding control);

биты 9..8 **PC** — управление точностью (precision control);

биты 7..6 зарезервированы;

Маскирование исключений

бит 5 **PM** — маска неточного результата;

бит 4 **UM** — маска антипереполнения;

бит 3 **OM** — маска переполнения;

бит 2 **ZM** — маска деления на 0;

бит 1 **DM** — маска денормализованного операнда;

бит 0 **IM** — маска недействительной операции.

Способы округления (RC — Round Control)

0 — к ближайшему целому;

1 — к отрицательной бесконечности;

2 — к положительной бесконечности;

3 — к нулю;

Точность результатов выполнения команд FADD, FSUB, FSUBR, FMUL, FDIV, FDIVR, FSQRT (PC — Precision Control).

0 — одинарная точность (32 битные числа);
1 — зарезервировано;
2 — двойная точность (64 битные числа);
3 — расширенная точность (80 битные числа);

Биты 0..5 маскируют соответствующие исключения — если бит установлен, исключение не инициируется, а результат выполнения команды, вызвавшей исключение, определяется правилами для каждого исключения отдельно.

Регистр тегов TW

Содержит 8 пар бит, описывающих содержимое каждого регистра данных:
15:14 — R7, …, 1:0 — R0.
Пара битов (тегов) означает:
11 — регистр пуст;
00 — регистр содержит число;
01 — регистр содержит 0;
10 — регистр содержит не число, бесконечность, ненормализованное число или неподдерживаемое значение;

Регистры FIP и FDP

Содержат адрес последней выполненной команды (за исключением команд **FINIT**, **FCLEX**, **FLDCW**, **FSTCW**, **FSTSW**, **FSTSWAX**, **FSTENV**, **FSAVE**, **FSTOR**, **FWAIT**) и адрес ее операнда.

Они используются в обработчиках исключений для анализа причин произошедшего.

Исключения FPU

Может возникать шесть типов исключений.

При возникновении исключения в регистре состояния **SR** устанавливается соответствующий флаг в 1 и, если маска этого исключения в регистре управления **CR** не установлена, вызывается:

- либо обычное прерывание **INT 10h**, если установлен бит **NE** в в регистре **CR0** центрального процессора,
- либо **IRQ13** (**INT 75h**), обработчик которого может прочитать регистры **SR**, **FIP** и **FDP**, чтобы определить тип исключения, породившую его команду и операнды.

В ряде случаев обработчик может исправить ситуацию.

Поведение в случае замаскированых исключений

Если бит маски наступившего исключения в регистре **CR** установлен, по умолчанию выполняются следующие действия:

P (неточный результат) — результат округляется в соответствии с битами RC (исключение происходит если, например, дробь 1/6 не может быть представлена вещественным числом любой точности и округляется).

При этом флаг **C1** показывает, в какую сторону произошло округление:
0 — вниз, 1 — вверх;

U (антипереполнение) — (исчезновение порядка) результат слишком мал, чтобы быть представленным в виде обычного числа — он преобразуется в денормализованное число;

O (переполнение) — результат преобразуется в бесконечность соответствующего знака;

Z (деление на ноль) — результат преобразуется в бесконечность соответствующего знака (учитывается знак нуля);

D (денормализованный операнд) — вычисления продолжаются как обычно;

I (недействительная операция) — результат определяется из следующей таблицы:

Операция	**Результат**
Ошибка стека	неопределенность
Операция с неподдерживаемым числом	неопределенность
Операция с NAN	QNAN
Сравнение числа с NAN	C0 = C2 = C3 = 1
Сложение бесконечностей с одним знаком или вычитание с разным	неопределенность
Умножение нуля на бесконечность	неопределенность
Деление бесконечности на бесконечность или 0/0	неопределенность
Команды FPREM и FPREM1, если делитель = 0 или делимое = бесконечности	неопределенность и C2 = 0
Тригонометрическая операция над бесконечностью	неопределенность и C2 = 0
Корень или логарифм, если x < 0, log(x+1) если x < -1	неопределенность
FBSTP, если регистр-источник пуст, содержит NAN, бесконеч-ность или превышает 18 десятичных знаков	десятичная неопределенность
FXCH, если один из операндов пуст	неопределенность

Арифметические и неарифметические инструкции

При работе с исключениями с плавающей запятой полезно различать арифметические инструкции и неарифметические инструкции.

Неарифметические инструкции не имеют операндов или не вносят существенных изменений в свои операнды.

Арифметические инструкции действительно вносят существенные изменения в свои операнды, в частности, они делают изменения, которые могут привести к выдаче сигналов об исключениях с плавающей запятой.

В ниже

Неарифметические и арифметические инструкции перечислены в таблице 8-9 «Arithmetic and Non-arithmetic Instructions» руководства программиста по архитектуре Intel 64 и IA-32[1].

Следует отметить, что некоторые неарифметические инструкции могут сигнализировать об исключении стека с плавающей запятой (квалифицируется как сбой), но это исключение не является результатом операции с операндом.

1 Intel® 64 and IA-32 Architectures Software Developer's Manual. Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4, Order Number: 325462-076US.

Условия возникновения исключений с плавающей запятой

Исключение недопустимой операции

Исключение недопустимой операции с плавающей запятой возникает в ответ на два подкласса операций:

- Переполнение или опустошение стека (**#IS**)
- Недействительный арифметический операнд (**#IA**)

Флаг этого исключения (**IE**) — это бит 0 слова состояния x87 FPU, а бит маски (**IM**) — это бит 0 слова управления x87 FPU.

Тип операции, вызвавшей исключение, указывается флагом ошибки стека (**SF**) слова в FSR.

Если флаг **SF** (флаг ошибки стека) установлен в 1, к переполнению или опустошению стека привела операция со стеком, если флаг сброшен, это недопустимый операнд.

FPU явно устанавливает флаг **SF** при обнаружении состояния переполнения или опустошения стека, но не очищает явно флаг при обнаружении условия недопустимого арифметического операнда.

В результате состояние флага **SF** может быть равно 1 после исключения недопустимой арифметической операции, если оно не было очищено с момента последнего возникновения переполнения или опустошения стека.

Исключение переполнения или опустошение стека (#IS)

Для отслеживания содержимого регистров в стеке регистров FPU используется слово тегов. Оно позволяет обнаружить два разных типа ошибок стека:

- переполнение стека — инструкция пытается загрузить данные из памяти в непустой регистр FPU. Непустой регистр — регистр, содержащий ноль (01), допустимое (00) или специальное значение (10).

- опустошение стека — инструкция ссылается на пустой регистр FPU как на операнд-источник, включая попытку записи содержимого пустого регистра в память. Пустой регистр имеет значение тега 11.

Термин «переполнение стека» происходит от случается, когда программа загрузила (поместила) восемь значений из памяти в стек регистров FPU. Следующее значение, помещаемое в стек, вызывает циклический переход стека к регистру, который уже содержит значение.

Термин «опустошение стека» случается в результате противоположной ситуации. В этом случае программа сохранила (извлекла) восемь значений из стека регистров FPU в память, и следующее значение, извлекаемое из стека, вызывает перенос указателя стека в пустой регистр.

Когда FPU обнаруживает переполнение или опустошение стека, он устанавливает флаг **IE** (бит 0) и флаг **SF** (бит 6) в слове состояния FPU в 1. Затем он устанавливает флаг кода условия **C1** (бит 9) в слове состояния FPU равным 1 в случае переполнения стека, или 0 в случае опустошения.

Исключение недопустимого арифметического операнда (#IA)

FPU x87 способен обнаруживать множество неверных арифметических операций, которые могут присутствовать в программе. Эти операции перечислены в таблице 8-10 документации Intel на IA-32. (Список включает недопустимые операции, определенные в стандарте IEEE 754.)

Когда FPU обнаруживает недопустимый арифметический операнд, он устанавливает в единицу флаг **IE** (бит 0) в слове состояния FPU.

Если исключение недопустимой операции замаскировано, FPU возвращает в операнде-адресате неопределенное значение с плавающей запятой, целое число или упакованное десятичное целое неопределенное значение или **QNaN** и/или устанавливает коды условий с плавающей запятой в зависимости от выполняемой инструкции, как показано в таблице. Это значение перезаписывает регистр назначения или ячейку памяти, указанную инструкцией.

Если исключение недопустимой операции не замаскировано, вызывается программный обработчик исключений, при этом указатель вершины стека (**TOP**) и исходные операнды остаются неизменными.

Обычно, когда один или оба исходных операнда являются **QNaN** (и ни один из них не является **SNaN** или находится в неподдерживаемом формате), исключение недопустимого операнда не возникает.

Исключением из этого правила является большинство инструкций сравнения (таких как инструкции **FCOM** и **FCOMI**) и инструкции преобразования чисел с плавающей запятой в целые числа (**FIST**/**FISTP** и **FBSTP**). С помощью этих инструкций исходный операнд **QNaN** сгенерирует исключение недопустимого операнда.

Исключение денормализованного операнда (#D)

FPU сигнализирует об исключении денормализованного операнда при следующих условиях:

- Если арифметическая инструкция пытается оперировать денормализованным операндом.
- При попытке загрузить денормализованное значение с плавающей запятой одинарной или двойной точности в регистр FPU.

Если загружаемое денормализованное значение является значением с плавающей запятой двойной расширенной точности, исключение денормализованного операнда не возникает.

Флаг (**DE**) для этого исключения — это бит 1 слова состояния FPU, а бит маски (**DM**) — это бит 1 управляющего слова FPU.

Когда возникает исключение денормализованного операнда и исключение замаскировано, FPU устанавливает флаг **DE**, а затем выполняет инструкцию.

Денормализованный операнд в формате с плавающей запятой одинарной или двойной точности при этом автоматически нормализуется при преобразовании в формат с плавающей запятой двойной расширенной точности. Последующие операции с этим значением будут выполняться с дополнительной точностью внутреннего формата с плавающей запятой двойной расширенной точности (80 бит).

Когда возникает исключение денормализованного операнда и исключение не замаскировано, устанавливается флаг **DE** и вызывается программный обработчик исключений. Указатель вершины стека (TOP) и исходные операнды остаются без изменений.

Исключение деления на ноль (#Z)

FPU сообщает об исключении деления на ноль с плавающей запятой всякий раз, когда инструкция пытается разделить конечный ненулевой операнд на 0.

Флаг (**ZE**) для этого исключения — это бит 2 слова состояния FPU, а бит маски (**ZM**) — это бит 2 управляющего слова FPU.

Команды **FDIV**, **FDIVP**, **FDIVR**, **FDIVRP**, **FIDIV** и **FIDIVR**, а также другие инструкции, выполняющие *внутреннее деление* (**FYL2X** и **FXTRACT**), так же могут сообщать об исключительной ситуации деления на ноль.

Когда возникает исключительная ситуация с делением на ноль и исключение замаскировано, FPU устанавливает флаг **ZE** и возвращает значения, показанные в следующей таблице[2] .

Условие	**Замаскированный ответ**
Операция деления или обратного деления с делителем 0	Возвращает в операнде-адресате ∞ со знаком исключающего ИЛИ знака двух операндов.
Инструкция FYL2X $y*\log_2(x+1)$	Возвращает в операнде-адресате ∞ со знаком, знак которой противоположен знаку ненулевого операнда
FXTRACT (Extract exponent and significand)	ST(1) устанавливается в −∞; ST(0) устанавливается в 0 с тем же самым знаком, что и операнд-источник

Если исключение деления на ноль не замаскировано, устанавливается флаг ZE, вызывается программный обработчик исключений (см. Раздел 8.7, «Обработка исключений x87 FPU в программном обеспечении») и указатель вершины стека (TOP) и исходные операнды остаются без изменений.

2 Таблица 8-10 Руководства программиста Order Number: 325462-074US.

Исключение числового переполнения (#O)

FPU сообщает об исключении числового переполнения с плавающей запятой (**#0**) всякий раз, когда округленный результат арифметической инструкции превышает наибольшее допустимое конечное значение, которое вписывается в формат с плавающей запятой операнда-адресата.

При использовании FPU числовое переполнение может происходить при арифметических операциях если результат сохраняется в регистре данных FPU.

Оно также может происходить при операциях сохранения с плавающей запятой (с использованием инструкций **FST** и **FSTP**), если значение в пределах диапазона в регистре данных сохраняется в памяти в формате с плавающей запятой одинарной или двойной точности.

Исключение числового переполнения не возникает при сохранении значений в целочисленном или целочисленном формате BCD.

Вместо этого сигнализируется исключение недопустимого арифметического операнда.

Флаг (**0E**) исключения числового переполнения — это бит 3 слова состояния FPU, а бит маски (**0M**) — это бит 3 управляющего слова FPU.

Когда возникает исключение числового переполнения и исключение замаскировано, FPU устанавливает флаг **0E** и возвращает округленное значение в соответствии с текущим режимом округления FPU.

Действие, которое выполняет FPU x87, когда происходит числовое переполнение и исключение числового переполнения не маскируется, зависит от того, должна ли инструкция сохранять результат в памяти или в стеке регистров:

Назначение ячейка памяти

В этом случае устанавливается флаг **OE** и вызывается программный обработчик исключений. Указатель вершины стека (TOP), а также исходный и целевой операнды остаются без изменений.

Поскольку данные в стеке имеют формат двойной расширенной точности, обработчик исключений имеет возможность либо повторно выполнить инструкцию сохранения после правильной настройки операнда, либо округлить значение в стеке до точности назначения, как того требует стандарт.

Если программа должна продолжаться, обработчик исключений должен в конечном итоге сохранить значение в целевом местоположении в памяти.

Назначение регистровый стек

Значение результата округляется в соответствии с текущими настройками битов управления точностью и округлением в управляющем слове FPU, а показатель степени результата корректируется путем деления его на 2^{24576}. Для инструкций, на которые не влияет поле точности, мантисса округляется до двойной точности. Результирующее значение сохраняется в операнде-адресате. Бит **C1** кода условия в слове состояния FPU (называемый в этой ситуации «битом округления») устанавливается в 1, если значение было округлено в большую сторону, и очищается, если результат был округлен в сторону 0. После того, как результат сохранен, устанавливается флаг **OE** и вызывается программный обработчик исключений. Так же корректируется экспонента.

При использовании инструкции **FSCALE** может произойти массивное переполнение, если результат будет слишком большим, чтобы его можно было представить, даже со скорректированным смещением экспоненты. В этом случае, если переполнение происходит снова, после смещения результата, в операнде-адресате сохраняется ∞ с правильным знаком.

Исключение потери числового значения (потери значимости) (#U)

FPU обнаруживает потенциальное состояние потери значимости числа с плавающей запятой всякий раз, когда результат арифметической инструкции отличен от нуля и является очень маленьким, то есть величина округленного результата не равна нулю и меньше наименьшего возможного нормализованного конечного значения, которое соответствует формату с плав. запятой в операнде-адресате.

Как и числовое переполнение, потеря значимости может происходить при арифметических операциях, как в случае сохранения результата в регистре данных FPU, так и в памяти (**FST** и **FSTP**), если значение в пределах диапазона в регистре данных сохраняется в памяти в меньших форматах с плавающей запятой одинарной или двойной точности. Исключение потери значимости не может возникнуть при сохранении значений в целочисленном формате или целочисленном формате BCD, поскольку значение с величиной меньше 1 всегда округляется до целого значения 0 или 1, в зависимости от действующего режима округления.

Флаг (**UE**) для исключения потери значимости — это бит 4 слова состояния FPU, а бит маски (**UM**) — это бит 4 управляющего слова FPU.

Когда возникает условие потери числового значения и исключение замаскировано, FPU выполняет операцию, описанную в Разделе 4.9.1.5, «Исключение потери числового значения (#U)» документа Intel на процессор IA-32.

Floating-Point Format	**Underflow Thresholds***
Single Precision	$\lvert x \rvert < 1.0 \cdot 2^{-126}$
Double Precision	$\lvert x \rvert < 1.0 \cdot 2^{-1022}$
Double Extended Precision	$\lvert x \rvert < 1.0 \cdot 2^{-16382}$

Исключение неточного результата (точности) (#P)

Исключение неточного результата (также называемое исключением точности) возникает, если результат операции не может быть точно представлен в целевом формате.

Все трансцендентные инструкции (**FSIN**, **FCOS**, **FSINCOS**, **FPTAN**, **FPATAN**, **F2XM1**, **FYL2X** и **FYL2XP1**) по своей природе дают неточные результаты.

Флаг исключения неточного результата (**PE**) — это бит 5 слова состояния FPU, а бит маски (**PM**) — это бит 5 слова управления FPU.

Если исключение неточного результата маскируется, то при возникновении условия неточного результата, числовое условие переполнения или потери значимости не возникает.

В слове состояния FPU устанавливается бит C1 (округление в большую сторону), чтобы указать, был ли неточный результат округлен в большую сторону (C1 == 1) или не был (C1 == 0).

В случае «без округления» младшие биты неточного результата усекаются так, чтобы результат соответствовал формату назначения.

Если исключение неточного результата не замаскировано, когда возникает неточный результат, но числовое переполнение или потеря значимости при этом не произошло, FPU обрабатывает исключение и вызывает программный обработчик исключений.

Синхронизация исключений FPU

Поскольку целочисленный блок и FPU являются отдельными исполнительными блоками, процессор может одновременно выполнять инструкции с плавающей запятой, целочисленные и системные инструкции. Для получения преимуществ одновременного выполнения не требуются специальные методы программирования — инструкции с плавающей запятой помещаются в поток инструкций вместе с целочисленными и системными инструкциями. Однако такое одновременное выполнение может вызывать проблемы с обработкой исключений с плавающей запятой.

Проблема связана с тем, как x87 FPU сигнализирует о существовании незамаскированных исключений с плавающей запятой.

Для замаскированных исключений с плавающей запятой специальной синхронизации исключений не требуется.

Если маска исключения с плавающей запятой сброшена и возникает условие исключения, FPU x87 останавливает дальнейшее выполнение инструкции с плавающей запятой и сигнализирует о событии исключения. При следующем появлении инструкции с плавающей запятой или инструкции **WAIT**/**FWAIT** в потоке инструкций процессор проверяет флаг **ES**[3] в слове состояния x87 FPU на наличие ожидающих исключений с плавающей запятой. Если такие существуют, FPU x87 делает неявный вызов (ловушка) программному обработчику исключений с плавающей запятой. После чего обработчик исключений может выполнять процедуры восстановления для выбранных или всех исключений с плавающей запятой.

3 ES — общий флаг ошибки. Если ES = 1, произошло хотя бы одно немаскированное исключение

Проблемы синхронизации возникают в промежутке времени между моментом сообщения об исключении и его фактической обработкой. Из-за одновременного выполнения в это время могут выполняться целочисленные или системные инструкции.

Таким образом, операнды источника или назначения для инструкции с плавающей запятой, в которой произошел сбой, могут быть перезаписаны в памяти, что делает невозможным для обработчика исключительной ситуации анализ или восстановление после исключения.

Чтобы решить эту проблему сразу после любой инструкции с плавающей запятой, которая может представлять ситуацию, когда информация о состоянии, относящаяся к исключению с плавающей запятой, может быть утеряна или повреждена, может быть размещена инструкция синхронизации исключения (либо инструкция с плавающей запятой, либо инструкция **WAIT**/**FWAIT**).

Основными кандидатами на синхронизацию являются инструкции с плавающей запятой, которые хранят данные в памяти. Например, следующие три строки кода могут вызвать проблемы с синхронизацией исключений:

```
FILD COUNT  ; Инструкция с плавающей запятой (загрузить целое в стек FPU)
INC  COUNT  ; Целочисленная инструкция
FSQRT       ; Следующая инструкция с плавающей запятой
```

В этом примере инструкция **INC** изменяет исходный операнд инструкции с плавающей запятой **FILD**. Если во время выполнения инструкции **FILD** поступил сигнал об исключении, инструкция **INC** могла бы перезаписать значение, хранящееся в ячейке памяти **COUNT**, до вызова обработчика исключений с плавающей запятой. После изменения переменной **COUNT** обработчик исключений с плавающей запятой не сможет исправить ошибку.

Но если инструкции перекомпоновать таким образом, чтобы инструкция **FSQRT** следовала за инструкцией **FILD**, что синхронизирует обработку исключений с плавающей запятой, возможность перезаписи переменной **COUNT** до вызова обработчика исключений с плавающей запятой будет исключена.

```
FILD COUNT  ; Инструкция с плавающей запятой (загрузить целое в стек FPU)
FSQRT       ; Следующая инструкция с плавающей запятой
INC  COUNT  ; Целочисленная инструкция
```

Инструкция **FSQRT** не требует какой-либо синхронизации, потому что результаты этой инструкции хранятся в регистрах данных x87 FPU и останутся там без изменений до тех пор, пока не будет выполнена следующая инструкция с плавающей запятой или **WAIT**/**FWAIT**. Чтобы абсолютно гарантировать, что любые исключения, исходящие из инструкции **FSQRT**, обрабатываются (например, до вызова процедуры), непосредственно после инструкции **FSQRT** может быть размещена инструкция **WAIT**.