

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(Самарский университет)

Борисов А.Н.

ВЫЧИСЛЕНИЯ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ

Методические указания к лабораторной работе 2

Самара, 2022

СОДЕРЖАНИЕ

Цели и задачи лабораторной работы	3
1 Введение	3
1.1 Исторические сведения	3
1.2 Стандарт IEEE-754	3
1.3 Ассемблер NASM и вещественные числа	6
2 Сопроцессор x87	7
2.1 Общая схема сопроцессора	7
2.2 Стек сопроцессора. Регистр тегов. Поле TOP регистра состояния	8
2.3 ПОЛИЗ	9
2.4 Регистр состояния. Управляющий регистр. Исключения	11
2.5 Инициализация сопроцессора. Сохранение состояния сопроцессора	13
2.6 Инструкции перемещения данных	13
2.7 Инструкции арифметики	15
2.8 Инструкции сравнения	16
2.9 Тригонометрические инструкции	17
2.10 Инструкции остатка, возведения в степень и логарифмирования	18
3 Набор инструкций SSE	20
3.1 Регистры XMM. Скалярный режим работы.	20
3.2 Инструкции перемещения и приведения типов	20
3.3 Инструкции арифметики	21
3.4 Инструкции сравнения	21
3.5 Регистр MXCSR. Маска исключений	22
3.6 Ограничения набора инструкций SSE	23
4 Задание на лабораторную работу	24
Приложение А. Пример установки режима округления	27
Приложение Б. Пример установки маски исключения	28
Приложение В. Примеры решения задач	29

Цели и задачи лабораторной работы

Цель лабораторной работы: изучение способов реализации вычислений с плавающей запятой на языке ассемблера NASM для архитектуры x86-64.

Задание на лабораторную работу: реализовать набор программ, решающих указанные математические задачи в 2-х вариантах: с помощью набора инструкций x87 и с помощью набора инструкций SSE.

1 Введение

1.1 Исторические сведения

Процессор Intel 8087, вышедший в 1978 г., мог осуществлять только целочисленные вычисления. Вычисления с плавающей запятой приходилось эмулировать с помощью целочисленных инструкций, что, очевидно, сказывалось на производительности.

В 1980 г. был выпущен математический сопроцессор (Floating Point Unit, FPU) Intel 8087, предназначенный для выполнения операций вещественной арифметики. Данный сопроцессор был выпущен в виде физически отдельного устройства, который работал параллельно с центральным процессором. Для работы с данным сопроцессором используются инструкции из набора, получившего название x87. Начиная с процессора Intel 80486 (1989 г.), сопроцессор был интегрирован в кристалл центрального процессора и перестал выпускаться в качестве отдельного устройства.

Сопроцессор выступал в качестве основного средства выполнения вещественных операций до 1999 г., в котором было выпущено расширение набора инструкций x86 под названием SSE (Streaming SIMD Extensions). Данный набор инструкций привнес 8 новых регистров XMM0-XMM7, которые позволяли выполнять арифметические операции над числами одинарной точности. В наборе инструкций SSE2 (2002 г.) добавились инструкции для работы с числами двойной точности, а число регистров XMM увеличилось до 16.

1.2 Стандарт IEEE-754

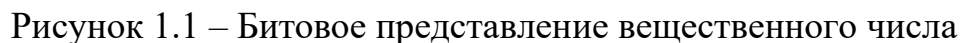
1.1.1 Представление чисел с плавающей запятой

Стандарт IEEE-754, принятый в 1985 г. регламентирует представление чисел с плавающей запятой, вычисления над ними и реакцию на некоторые особые случаи, возникающие при вычислениях.

Согласно данному стандарту, числа с плавающей запятой могут кодироваться 32 (числа одинарной точности) либо 64 битами (числа

На аппаратном уровне числа представляются в виде $(-1)^s \cdot m \cdot 2^{exp-bias}$, где s – знак числа (0 – положительное, 1 – отрицательное), m – мантисса, exp – экспонента. Мантисса и экспонента кодируются в виде *положительных* чисел, при этом значения $exp = 0$ и $exp = 11 \dots 11_2$ является зарезервированным.

На битовом уровне число кодируется, как показано на рисунке 1.1 с пояснениями в таблице 1.1.



Точность числа	M	E	bias
Одинарная	24	8	127
Двойная	53	11	1023

1.1.2 Бесконечность, денормализованные числа и не-числа

Особыми случаями являются денормализованные числа, бесконечность и не-числа.

Денормализованные числа идентифицируются по значению $exp = 0$. Для денормализованных чисел первый (неявный) бит равен 0. Итоговое значение денормализованного числа представимо в виде $\pm 0, m_0 m_1 \dots \times 2^{-bias+1}$ (отсюда следует, что ± 0 – особое денормализованное число), где $m_0 m_1 \dots$ – биты мантиссы. Модуль

денормализованного числа меньше модуля любого нормализованного числа.

В зависимости от настроек, при получении денормализованного числа в результате операции FPU может просто вернуть результат, округлить его до 0, или выдать аппаратное исключение (Underflow Exception, исключение потери точности).

Остальные особые случаи идентифицируются значением $exp = 2^E - 1$ (т.е. все поле экспоненты заполнено 1).

Бесконечность (как положительная, так и отрицательная), идентифицируются по мантиссе = 0.

Не-числа (Not-a-Number, NaN) идентифицируются по мантиссе $\neq 0$. Не-числа являются результатом запрещенной операции ($\ln(-1)$, $\frac{\infty}{\infty}$, $0 \times \infty$, $\mp\infty + \pm\infty$, $\pm\infty - \pm\infty$ и др.). Если хоть один из операндов является NaN, результатом любой операции также является NaN. *NaN не равно никакому другому числу (включая само NaN).*

Сигнальные NaN, или sNaN, (signaling NaN, мантисса в виде $0\dots_2 \neq 0$) заставляют FPU выпустить аппаратное исключение при выполнении операции над ними.

«Тихие» NaN, или qNaN, (quiet NaN, мантисса в виде $1\dots_2$) не заставляют FPU выпускать исключения. Обычно qNaN являются результатом работы FPU при отключенных аппаратных исключениях.

Для обоих типов NaN оставшаяся часть мантиссы (все биты, кроме первого) может содержать любое значение, и может использоваться для передачи дополнительной информации (payload).

Итоговый вид битовых представлений различных типов вещественных чисел приведен в таблице 1.2 (знаковый бит опущен, т.к. не учитывается).

Таблица 1.2. Битовые представления типов вещественных чисел.

Тип числа	Экспонента	Мантисса
Нормализованное	$00\dots_2 01_2 - 11\dots_2 10_2$	$00\dots_2 00_2 - 11\dots_2 11_2$
Денормализованное	$00\dots_2 00_2$	$00\dots_2 00_2 - 11\dots_2 11_2$
Бесконечность	$11\dots_2 11_2$	$00\dots_2 00_2$
sNaN	$11\dots_2 11_2$	$00\dots_2 01_2 - 01\dots_2 11_2$
qNaN	$11\dots_2 11_2$	$10\dots_2 00_2 - 11\dots_2 11_2$

1.1.3 Режимы округления

Поскольку в ходе вычислений с плавающей запятой неизбежно возникают округления в младшем разряде, в стандарте IEEE-754 определены 4 возможных режима округления:

- 1) округление к ближайшему целому;
- 2) **округление вверх** ($\kappa + \infty$);
- 3) **округление вниз** ($\kappa - \infty$);
- 4) округление в сторону 0 (к минимальному модулю).

Данные режимы в том числе применяются при преобразовании вещественных чисел в целые.

Результаты округления числа 0,3:

- 1) к ближайшему целому: 0,0
- 2) **округление вверх** ($\kappa + \infty$): 1,0
- 3) **округление вниз** ($\kappa - \infty$): 0,0
- 4) округление в сторону 0: 0,0

Результаты округления числа -0,7:

- 1) к ближайшему целому: -1,0
- 2) **округление вверх** ($\kappa + \infty$): -0,0
- 3) **округление вниз** ($\kappa - \infty$): -1,0
- 4) округление в сторону 0: -0,0

1.3 Ассемблер NASM и вещественные числа

В языке ассемблера NASM вещественные числа кодируются на уровне их байтовых представлений.

В секциях `.data/.rodata` объявить вещественное число можно следующим образом:

```
section .data
    some_float: dd 0.0
    some_double: dq -1.0
    some_long_double: dt 255.21
```

Вещественная константа будет закодирована согласно размеру ячейки (dd – одинарная точность, dq – двойная точность, dt – расширенная точность). Псевдоинструкция dt выделяет 10 байт.

Для помещения специальных значений в NASM определены константы `__Infinity__`, `__QNaN__` и `__SNaN__`:

```
float_inf: dd __Infinity__  
double_qnan: dq __QNaN__  
long_double_snan: dt __SNaN__
```

Для записи в память или в регистр общего назначения вещественного числа используются макросы `__float32__` и `__float64__`:

```
mov dword[x], __float32__(-1.0)  
mov rax, __float64__(0.1)
```

Данные макросы вычисляют битовое представление вещественного числа, а затем подставляют его в виде целого числа в инструкцию. Реально исполняются следующие инструкции:

```
mov dword[x], 0xbf800000  
mov rax, 0x3fb999999999999a
```

2 Сопроцессор x87

2.1 Общая схема сопроцессора

Сопроцессор x87 реализован по схеме стековой машины. Отличительной особенностью сопроцессора является то, что по умолчанию он работает не по стандарту IEEE-754. Операции в сопроцессоре выполняются с точностью **80-бит** (тип `long double` в языке C, некоторые компиляторы, в частности MSVC, считают `long double` = `double`).

Общая схема сопроцессора приведена на рисунке 3.1.

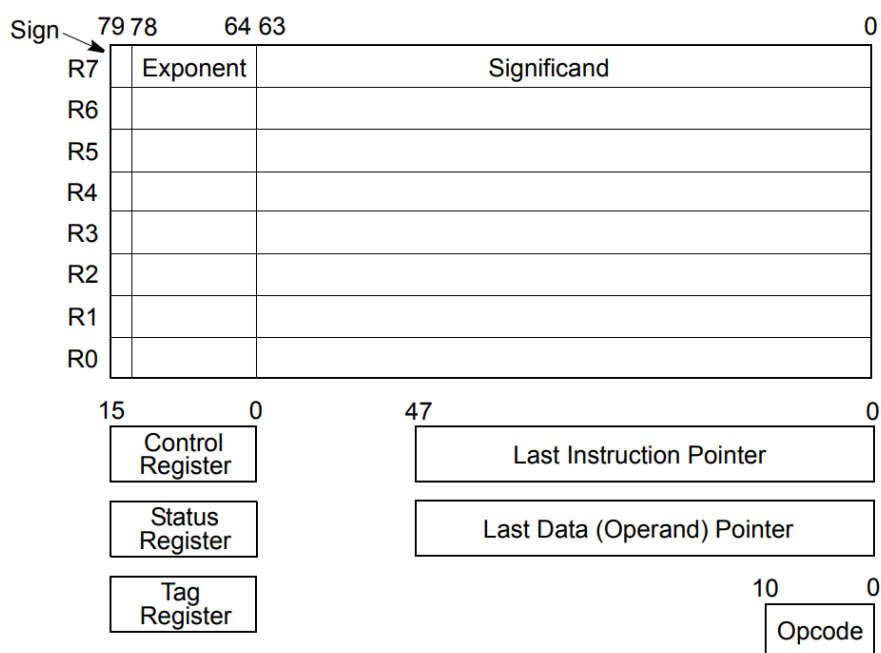


Рисунок 2.1 – Общая схема сопроцессора x87

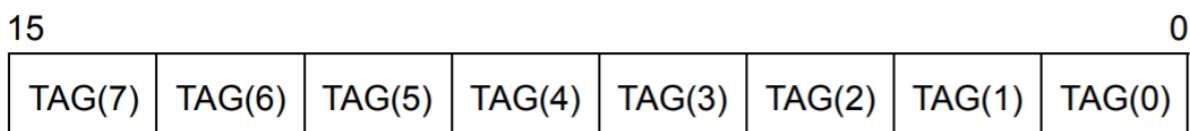
Сопроцессор имеет 8 регистров данных (R0-R7 на рисунке), организованных в виде стека, управляющий регистр (Control Register на рисунке), регистры состояния и тэгов (Status Register и Tag Register на рисунке соответственно), а также регистры последней инструкции, операндов и опкода.

2.2 Стек сопроцессора. Регистр тегов. Поле TOP регистра состояния

В отличие от регистров общего назначения, регистры x87 не доступны по своим исходным именам. Регистры сопроцессора организованы в стек. В рамках данного стека регистры могут быть либо заняты, либо пустыми. Чтение или запись в пустую ячейку приводят к аппаратному исключению.

Индекс регистра, являющегося вершиной текущего стека, хранится в поле TOP регистра состояния. Регистры адресуются относительно текущей вершины. К примеру, регистр `st0` соответствует текущей вершине стека, регистр `st2` – третьему относительно вершины регистру. Аналогично стеку вызовов, стек сопроцессора растет вниз.

Текущее состояние каждого регистра хранится в регистре тегов, структура которого приведена на рисунке 2.2.



TAG Values

00 — Valid

01 — Zero

10 — Special: invalid (NaN, unsupported), infinity, or denormal

11 — Empty

Рисунок 2.2 – Регистр тегов

Регистр тегов разделен на 8 частей по 2 бита, каждая из которых хранит состояние того или иного регистра. Значение 11 в поле тега означает, что регистр не содержит значения (находится вне стека), чтение из этого регистра приведет к ошибке. Остальные значения означают, что регистр является частью текущего стека, и содержит либо нормализованное число (тег 00), либо 0 (тег 01), либо иное ненормализованное число (тег 10).

Пример состояния стека с 3 числами приведен на рисунке 2.3.

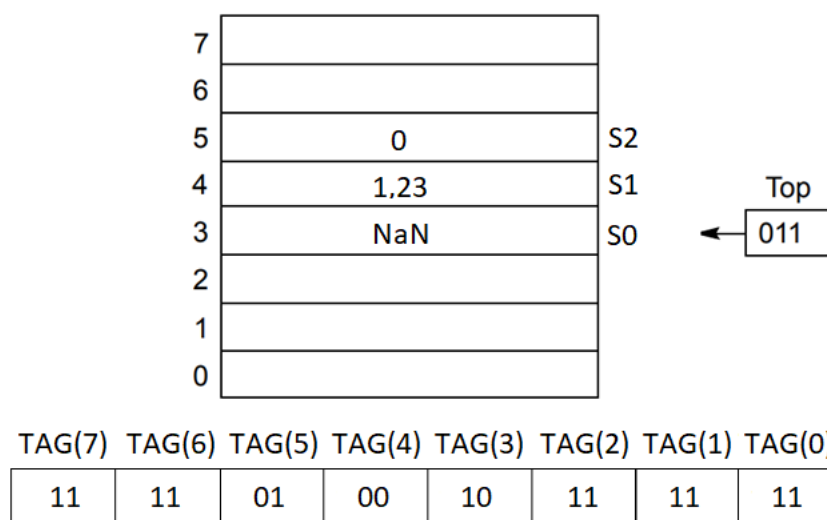


Рисунок 2.3 – Пример стека FPU с 3 элементами

2.3 ПОЛИЗ

Ответ на вопрос о том, почему регистры сопроцессора организованы в стек, можно легко получить, если узнать, каким образом можно кодировать последовательность вычислений математического выражения.

Стандартный способ записи математических выражений (инфиксная запись) обладает неоднозначностью, которую приходится решать с

помощью управляющих символов (скобок). К примеру, $a+b/c$ и $(a+b)/c$ являются совершенно разными выражениями

Префиксная и постфиксная запись не обладают данными недостатками. Префиксная запись была изобретена польским математиком Яном Лукасевичем в 1920 г, и получила название польской нотации или польской записи. Постфиксная запись получила в информатике большее распространение, и в настоящее время широко известна под названиями польской обратной записи, польской обратной нотации, польской инверсной записи или ПОЛИЗ.

В ПОЛИЗ оператор записывается после операндов. К примеру, $a+b$ будет записано, как $a\ b\ +$. Выражения $a+b/c$ и $(a+b)/c$ будут записаны, как $a\ b\ c\ /\ +$ и $a\ b\ +\ c\ /\$.

Замечательными свойствами данной нотации является то, что она не имеет неоднозначностей и то, что она допускает простую реализацию вычислений на основе стековой машины по следующему правилу:

- 1) Выражение читается слева направо.
- 2) Если текущий элемент – операнд, поместить в стек.
- 3) Если текущий элемент – оператор, прочитав из стека требуемое количество операндов, обратить их порядок, выполнить операцию, записать результат в стек.
- 4) Если вся строка прочитана, то результат – на вершине стека.

К примеру, для $3\ 16\ 8\ /\ +$, на момент выполнения операции $/$ стек будет иметь вид $[8\ 16\ 3]$, из стека будут взяты $8\ 16$ (в таком порядке), операция будет выполнена, как $16/8=2$ (отметьте изменение порядка), после операции стек будет содержать $[2\ 3]$. Процесс выполнения $+$ тривиален, после операции стек будет содержать $[5]$.

Для $3\ 16\ +\ 8\ /\$, на момент выполнения операции $+$ стек будет иметь вид $[3\ 16]$, из стека будут взяты $3\ 16$, операция будет выполнена, как $16+3=19$, после операции стек будет иметь вид $[19]$. На момент выполнения операции $/$ стек будет иметь вид $[8\ 19]$, из стека будут взяты $8\ 19$, операция будет выполнена, как $19/8=2,375$, после операции стек будет иметь вид $[2,375]$.

Во время компиляции математические выражения преобразуются в ПОЛИЗ-запись, которая потом отлично транслируется в инструкции сопроцессора. Алгоритм Дейкстры для преобразования инфиксной записи в постфиксную можно легко найти в открытых источниках.

2.4 Регистр состояния. Управляющий регистр. Исключения

В процессе выполнения арифметических операций могут возникнуть различного рода ошибки: деление на 0, потеря точности и пр.

Аналогично регистру флагов RFLAGS, результат последней операции отражается в регистре состояния сопроцессора. Структура регистра приведена на рисунке 2.4.

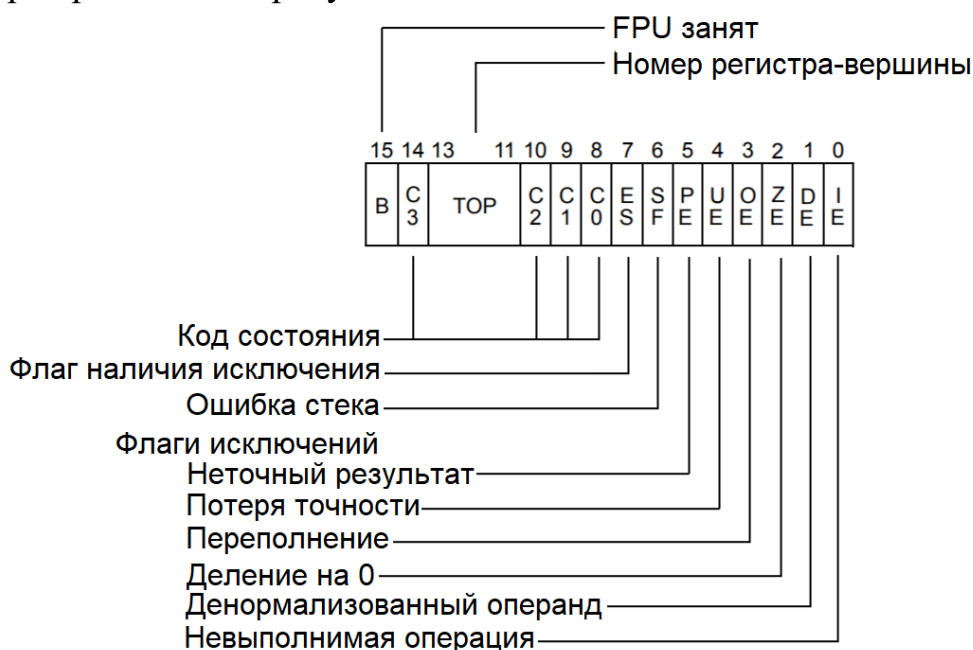


Рисунок 2.4 – Регистр состояния FPU

Код состояния определяет результат последней операции. Ранее код состояния использовался при реализации сравнений, однако начиная с процессоров архитектуры Р6 данные коды более могут не использоваться.

Флаг *ошибки стека* происходит при переполнении стека (попытке добавить элемент в уже полный стек) либо при попытке удаления элемента из пустого стека.

Флаг наличия исключения равен 1, если равен 1 любой из флагов исключений. Флаги исключений сигнализируют об ошибке, произошедшей в результате последней операции.

Ошибка неточного результата возникает, если результат операции не может быть представлен в точности (например, бесконечная дробь $1/3$ не может быть представлена точно).

Ошибка потери точности возникает, если результатом операции является денормализованное число.

Ошибка переполнения возникает, если результатом операции является бесконечность.

Ошибка денормализованного операнда возникает, если один из операндов является денормализованным числом.

Ошибка невыполнимой операции возникает, если запрошенная операция не может быть выполнена корректно (например, $\ln -1$).

Любая ошибка приводит к аппаратному исключению, если соответствующий флаг в маске исключений не равен 1. Маска исключений хранится в управляющем регистре (рисунок 2.5)

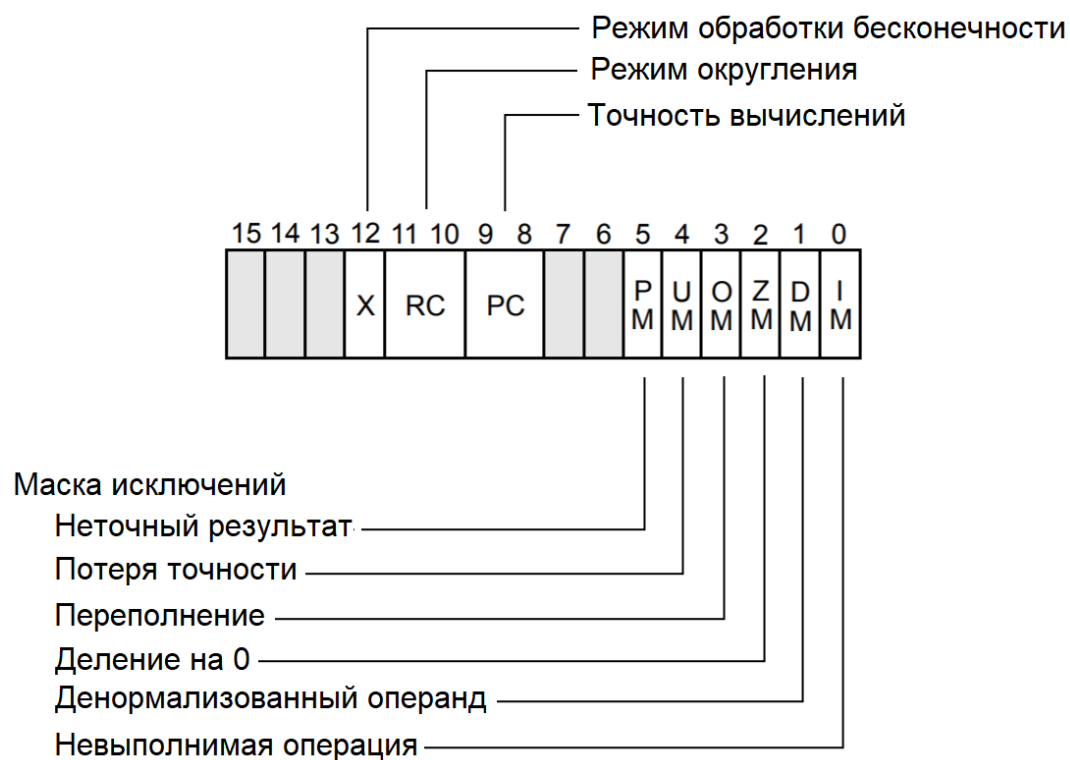


Рисунок 2.5 – Управляющий регистр FPU

Помимо маски состояния, управляющий регистр также содержит 2 бита, указывающие режим округления (см. таблицу 3.1) и 2 бита, указывающие точность вычислений (в пределах л/р не используются).

Сохранение регистра состояния производится инструкциями FSTSW/FNSTSW (FPU STore Status Word). Инструкция FSTSW проверяет наличие немаскированных исключений перед сохранением слова состояния в памяти, инструкция FNSTSW сохраняет слово состояния без проверок. Сохранение слова состояния может потребоваться для проверки кода состояния или его отдельных флагов C0–C3.

Сохранение управляющего регистра производится инструкциями FSTCW/FNSTCW (FPU STore Control Word). Инструкция FSTCW проверяет наличие немаскированных исключений перед сохранением слова состояния в памяти, инструкция FNSTCW сохраняет значение управляющего регистра без проверок.

Загрузка управляющего регистра производится инструкцией FLDCW (FPU Load Control Word).

Отключить возникновение аппаратных исключений можно последовательностью инструкций

```
FNSTCW [addr]
MOV byte[addr], 0x3F
FLDCW [addr]
```

Хотя ошибки детектируются FPU немедленно, аппаратные исключения, по историческим причинам, бросаются только во время выполнения *следующей* инструкции FPU (т.е., ошибка деления на 0 при FDIVP может быть обнаружена только во время выполнения следующей инструкции, например FADD).

Для немедленной проверки на наличие немаскированных исключений используется инструкция FWAIT. В то время, когда сопроцессор был отдельным устройством, данная инструкция использовалась для ожидания окончания выполнения FPU текущей операции. В настоящее время ее единственная функция – проверка на исключения. Многие инструкции сопроцессора, проверяющие наличие немаскированных исключений, внутри используют FWAIT, например FSTCW=FWAIT+FNSTCW.

2.5 Инициализация сопроцессора. Сохранение состояния сопроцессора

Для того, чтобы реинициализировать сопроцессор (сбросить все значения, вернуть управляющий регистр и регистр состояния к начальным значениям) используются инструкции FINIT/FNINIT (первая предварительно проверяет наличие немаскированных исключений).

После реинициализации *все исключения по умолчанию маскированы*.

Для того, чтобы сохранить полное состояние сопроцессора, используются инструкции FSAVE/FNSAVE (различия те же). Данные инструкции сохраняют полное состояние сопроцессора (108 байт). Для загрузки сохраненного состояния используется инструкция FRSTOR (FPU ReStORe).

2.6 Инструкции перемещения данных

Поскольку регистры сопроцессора организованы в виде стека, загрузка данных в стек и выгрузка данных из него возможны только в/из вершины стека.

Поместить новое вещественное число в стек можно инструкцией FLD. При этом стек увеличивается на 1. Данная инструкция принимает только 1 аргумент – источник, в роли которого может выступать другой регистр сопроцессора или память.

Примеры:

FLD ST5 ; поместить содержимое регистра ST5 в стек (ST5 станет при этом ST6)

FLD dword[rax] ; поместить число одинарной точности в стек.

FLD qword[rbx] ; поместить число двойной точности в стек.

FLD tbyte[rcx] ; поместить число расширенной точности в стек

При загрузке данных в сопроцессор вещественное число автоматически приводится к 80-битному формату.

Выгрузка вещественных чисел осуществляется инструкциями FST и FSTP. Инструкция FST сохраняет значение на вершине стека. Инструкция FSTP дополнительно выталкивает значение из вершины стека (т.е. размер стека уменьшается).

Примеры:

FST ST5 ; ST5=ST0

FSTP ST5 ; ST5=ST0, ST0 выталкивается, ST5 становится ST4.

FST qword[rbx] ; сохранить в память число одинарной точности

FSTP qword[rbx] ; сохранить в память число двойной точности, вытолкнуть значение из вершины стека

Загрузка знаковых целых чисел с последующим преобразованием в вещественное число производится инструкцией FILD. Данная инструкция принимает единственный аргумент, который должен быть адресом (не целочисленным регистром!). При этом указатель на byte не поддерживается, только word/dword/qword.

Примеры:

FILD word[rax] ; преобразование short -> long double.

FILD dword[rax] ; преобразование int -> long double.

FILD qword[rax] ; преобразование long long -> long double.

Преобразование вещественных чисел в целое производится инструкциями FIST и FISTP. FISTP дополнительно выталкивает значение из вершины стека. Преобразование в qword поддерживается только инструкцией FISTP.

Примеры:

FIST word[rax] ; преобразование long double->short.

FISTP qword[rbx] ; преобразование long double->int с выталкиванием из вершины.

Обмен значений между вершиной стека и другим регистром производится инструкцией FXCH. Данная инструкция принимает 1 операнд – регистр, с которым вершина стека обменивается значением.

Пример: FXCH ST2 ; ST2 \leftrightarrow ST0

Для загрузки констант в вершину стека используется ряд специальных инструкций:

FLD1 ; поместить 1 в стек.

FLDZ ; поместить 0 в стек.

FLDPI ; поместить π в стек.

FLDL2T ; поместить 10 в стек.

FLDL2E ; поместить e в стек.

FLDLG2 ; поместить 2 в стек.

FLDLN2 ; поместить 2 в стек.

2.7 Инструкции арифметики

Список инструкций, выполняющих стандартные арифметические операции, приведен ниже:

FADD/FADDP ; сложение

FMUL/FMULP ; умножение

FSUB/FSUBR/FSUBP/FSUBR ; вычитание

FDIV/FDIVR/FDIVP/FDIVR ; деление

Поскольку арифметические операции выполняются одинаково, далее будут рассмотрена только операция вычитания.

Инструкция FSUB может принимать 0,1 или 2 операнда.

В случае отсутствия операндов инструкции FSUB и FSUBP являются синонимами – операция выполняется над ST1 и ST0, результат записывается в ST1, значение из вершины стека выталкивается.

В случае одного операнда первым неявным операндом является ST0. Значение аргумента вычитается из ST0 без выталкивания.

В случае 2-х операндов один из операндов должен быть регистр ST0 – вершина стека. Выталкивания значения из вершины не происходит.

Примеры:

FSUB ; ST1=ST0, вытолкнуть значение из вершины.

FSUB qword[rax] ; ST0=(long double)*RAX.

FSUB st5 ; ST0=ST5.

FSUB st0, st5 ; ST0=ST5.

FSUB st5, st0 ; ST5=ST0.

Инструкция FSUBR действует аналогично, но при этом порядок элементов при вычитании меняется.

FSUBR ; ST1=ST0-ST1, вытолкнуть значение из вершины.

FSUBR qword[rax] ; ST0=(long double)*RAX-ST0.

FSUBR st5 ; ST0=ST5-ST0.

FSUBR st0, st5 ; ST0=ST5-ST0.

FSUBR st5, st0 ; ST5=ST0-ST5.

Инструкции FSUBP и FSUBRP выполняются аналогично своим аналогам без суффикса P, но при этом всегда выталкивают значение с вершины стека.

Приведенные в разделе 3.3 примеры могут быть реализованы следующим образом (считая, что все аргументы – числа одинарной точности).

Выражение: $x = a + b / c$

ПОЛИЗ: a b c / +

Ассемблер:

```
FLD dword[a]
FLD dword[b]
FLD dword[c]
FDIV
FADD
FSTP dword[x]
```

Выражение: $x = (a + b) / c$

ПОЛИЗ: a b + c /

Ассемблер:

```
FLD dword[a]
FLD dword[b]
FADD
FLD dword[c]
FDIV
FSTP dword[x]
```

2.8 Инструкции сравнения

Сравнение реализуется с помощью инструкций FCOMI и FCOMIP (FPU COMpare and set Integer flags). Данные инструкции принимают регистр FPU в качестве операнда и сравнивают его с вершиной стека. FCOMIP

дополнительно выталкивает значение из вершины стека.
Пример: FCOMI st1.

Результат сравнения записывается в флаги ZF, CF и PF регистра RFLAGS согласно таблице 3.1.

Таблица 3.1. Эффект операции вещественного сравнения на RFLAGS

Результат сравнения	ZF	PF	CF
ST0 > ST(i)	0	0	0
ST0 < ST(i)	0	0	1
ST0 = ST(i)	1	0	0
Нет порядка (NaN)	1	1	1

Поскольку результат сравнения записывается в флаг CF, следует использовать условные переходы, завязанные на беззнаковое сравнение (JA, JB)

Если исключение невыполнимой операции не маскировано, то FCOMI сгенерирует аппаратное исключение, если хотя бы один из операндов является NaN. Для корректной обработки NaN необходимо либо маскировать исключение, либо использовать инструкции FUCOMI/FUCOMIP.

2.9 Тригонометрические инструкции

Тригонометрические функции им вычисляются следующими инструкциями (значения интерпретируются в радианах):

FSIN ; ST0 = sin(ST0)

FCOS ; ST0 = cos(ST0)

FSINCOS ; X = sin(ST0), Y = cos(ST0); вытолкнуть из стека вершину, поместить в стек Y и X.

FPTAN ; ST0 = tg (ST0), затем поместить в стек 1.0.

FPATAN ; ST1 = arctg(ST1/ST0), вытолкнуть из стека вершину.

Модуль экспоненты аргумента должен быть не более 63 (т.е модуль числа должен находиться в диапазоне (2^{-64} ; 2^{64})). О причинах подобного см. инструкцию FPREM1.

Инструкции для прямого вычисления котангенса, арксинуса, арккосинуса и арккотангенса отсутствуют.

2.10 Инструкции остатка, возведения в степень и логарифмирования

Вещественный остаток от деления на число рассчитывается инструкциями FPREM и FPREM1. FPREM1 рассчитывает остаток, согласно IEEE-754.

Данные инструкции рассчитывают остаток от деления ST0 на ST1. При этом, если разница в экспоненте между данными числами более 63 (т.е. остаток от деления ST0 на ST1 по модулю больше 2^{63}), то инструкция устанавливает флаг C2=1 в слове состояния и возвращает частичный остаток. Как следствие, если нет гарантии, что ST0 и ST1 отличаются менее, чем в 2^{63} раз, инструкции FPREM/FPREM1 должны выполняться в цикле с выгрузкой слова состояния инструкцией FSTSW в память и явной проверкой флага C2.

Вычисление квадратного корня выполняется инструкцией FSQRT. Данная инструкция вычисляет $ST0 = \sqrt{ST0}$.

Вычисление логарифма выполняется с помощью инструкции FYL2X. Данная инструкция вычисляет $ST1 = ST1 * \log_2(ST0)$ с вытеснением вершины из стека. Данная инструкция позволяет считать логарифм по другому основанию по формуле $x = \frac{x}{a}$.

Возведение в степень производится с помощью инструкций F2XM1, FSCALE и FPREM.

Инструкция F2XM1 вычисляет $ST0 = 2^{ST0} - 1$. ST0 должно быть по модулю < 1 .

Инструкция FSCALE вычисляет $ST1 = ST1 * 2^{[ST0]}$ и вытесняет значение из стека (возводит 2 в степень, равную ближайшему целому к ST0).

Возведение в степень по основанию, отличному от 2, производится по формуле $a^b = 2^{b \cdot a} - 1 + 1$. Показатель степени в данном выражении можно вычислить инструкцией FYL2X, дробная часть показателя извлекается инструкцией FPREM, далее целая и дробная части показателя используются инструкциями FSCALE и F2XM1. Итоговый код на языке ассемблера:

```
FLD dword[b]
FLD dword[a]
FYL2X      ;вычисляем показатель
FLD1       ;загружаем +1.0 в стек
FLD ST1    ;дублируем показатель в стек
FPREM      ;получаем дробную часть
F2XM1      ;возводим в дробную часть показателя
```

```
FADD      ;прибавляем 1 из стека
FSCALE    ;возводим в целую часть и умножаем
FSTP ST1  ; выталкиваем лишнее из вершины
```

3 Набор инструкций SSE

3.1 Регистры XMM. Скалярный режим работы.

Набор инструкций SSE2 включает в себя 16 регистров, обозначаемых XMM0-15. Регистры XMM независимы друг от друга, в отличие от регистров сопроцессора.

XMM-регистры являются векторными регистрами, их размер – 16 байт. В скалярном режиме используется только младшие 4 или 8 байт регистра. Для работы с числами одинарной точности используются инструкции с суффиксом SS (Scalar Single, пример – ADDSS). Для работы с числами двойной точности используется инструкции с суффиксом SD (Scalar Double, пример - ADDSD).

3.2 Инструкции перемещения и приведения типов

Перемещение данных производится инструкциями MOV S_x (где вместо x – S для чисел одинарной точности, D для чисел двойной точности). Инструкции принимают 2 аргумента – источник и приемник. Хотя бы один аргумент должен быть XMM-регістром. Если аргументом является указатель, то размер читаемых/записываемых данных соответствует типу данных инструкции (4 или 8 байт)

Примеры:

```
MOVSS xmm0, [rax]
MOVSD [rbx], xmm1
MOVSS xmm1, xmm0
```

Переместить значение без приведения типов из целочисленного регистра в XMM-регистр можно инструкциями MOVD/MOVQ, суффикс указывает на размер данных (4/8 байт). Данные инструкции полезны для загрузки констант при помощи макросов из раздела 1.3. Пример:

```
MOV RAX, __float64__(-1.0)
MOVQ XMM0, RAX ;
MOV RCX, __float32__(128.23)
MOVD XMM1, ECX ;
```

Приведение между числами разной точности производится инструкциями CVTSS2SD и CVTSD2SS. Инструкция CVTSS2SD (ConVerT Scalar Single To Scalar Double) выполняет приведение float=>double (одинарную в двойную точность). Инструкция CVTSD2SS (ConVerT Scalar Double To Scalar Single) выполняет приведение double=>float (двойную в

одинарную точность). Аргументы аналогичны инструкциям MOVSS/MOVSF.

Преобразование целых чисел в вещественные производится инструкциями CVTSS2SI и CVTSD2SI (где вместо x – S или D) (ConVerT Scalar Integer To ...). Инструкции CVTSS2SI преобразуют целое число в число одинарной/двойной точности. Инструкции CVTSD2SI преобразуют число одинарной/двойной точности в целое. Для инструкций CVTSD2SI приемник должен быть регистром общего назначения. Источник для всех инструкций может быть как регистром, так и указателем. Размер целого числа выводится из размера аргумента.

Примеры:

```
CVTSS2SI xmm1, eax ; xmm1 = (float)eax
CVTSS2SD xmm2, [rax]; xmm2 = (double)(*rax)
CVTSD2SI eax, xmm3 ; eax=(int)xmm3
CVTSD2SI rax, [rbx] ; rax=(long long)(*rbx)
```

3.3 Инструкции арифметики

Стандартные инструкции арифметики выполняются следующими инструкциями (где x – S или D):

```
ADDSx XMM, ARG ; XMM+=ARG.
SUBSx XMM, ARG ; XMM-=ARG.
DIVSx XMM, ARG ; XMM/=ARG.
MULSx XMM, ARG ; XMM*=ARG.
SQRTSx XMM, ARG ; XMM=sqrt(ARG)
RCPSx XMM, ARG ; XMM=1/ARG
RSQRTSS XMM, ARG ; XMM=1/sqrt(ARG)
```

Все инструкции принимают 2 операнда. Первый операнд – всегда XMM-регистр, второй операнд может быть XMM-регистром или указателем.

Примеры:

```
ADDSS xmm1, xmm2
SUBSD xmm1, [rax]
SQRTSS xmm2, [rbx]
RCPSD xmm2, xmm3
```

3.4 Инструкции сравнения

Если сравнение необходимо для дальнейшего выполнения условного перехода, то следует использовать инструкции COMISx или UCOMISx. Инструкции сравнивают операнды и выставляют флаги ZF, CF и PF

регистра RFLAGS согласно таблице 3.1. Инструкция UCOMISx используется, если NaN является допустимым значением.

Пример:

```
COMISS xmm1, [rax]
JZ label
```

Для сравнения вещественных чисел одинарной и двойной точности также используются инструкции CMPSx (CoMPare Scalar Single/Double). Это трехоперандные инструкции, третий аргумент которых является целочисленной константой, указывающей тип сравнения. Данные инструкции заполняют регистр-приемник битами 1 в случае, если результатом сравнения является True, и 0 – если False.

Для удобства в языке ассемблера NASM существуют специальные псевдоинструкции, реализующие сравнение чисел с использованием CMPSx:

```
CMPEQSx XMM, ARG ; XMM = XMM==ARG ? -1 : 0
CMPLTSx XMM, ARG ; XMM = XMM<ARG ? -1 : 0
CMPLESx XMM, ARG ; XMM = XMM<=ARG ? -1 : 0
CMPNEQSx XMM, ARG ; XMM = XMM!=ARG ? -1 : 0
CMPNLTSx XMM, ARG ; XMM = XMM>=ARG ? -1 : 0
CMPNLESx XMM, ARG ; XMM = XMM>ARG ? -1 : 0
CMPUNORDSx XMM, ARG ; XMM=(XMM is NaN)|| (ARG is NaN)?-1:0
CMPORDSx XMM, ARG ; XMM=!((XMM is NaN)|| (ARG is NaN)) ?-1:0
```

Если необходимо определить наибольшее из 2-х чисел, то вместо инструкций сравнения следует использовать инструкции MINSx/MAXSx.

```
MINSx XMM, ARG ; XMM = XMM<ARG ? XMM : ARG
MAXSx XMM, ARG ; XMM = XMM>ARG ? XMM : ARG
```

3.5 Регистр MXCSR. Маска исключений

Аналогично сопроцессору, в наборе инструкций SSE существует регистр для хранения управляющих флагов и слова состояния SSE-FPU. В отличие от сопроцессора x87, вся информация хранится в одном регистре MXCSR (MX Control and Status Register). Схема регистра приведена на рисунке 4.1.

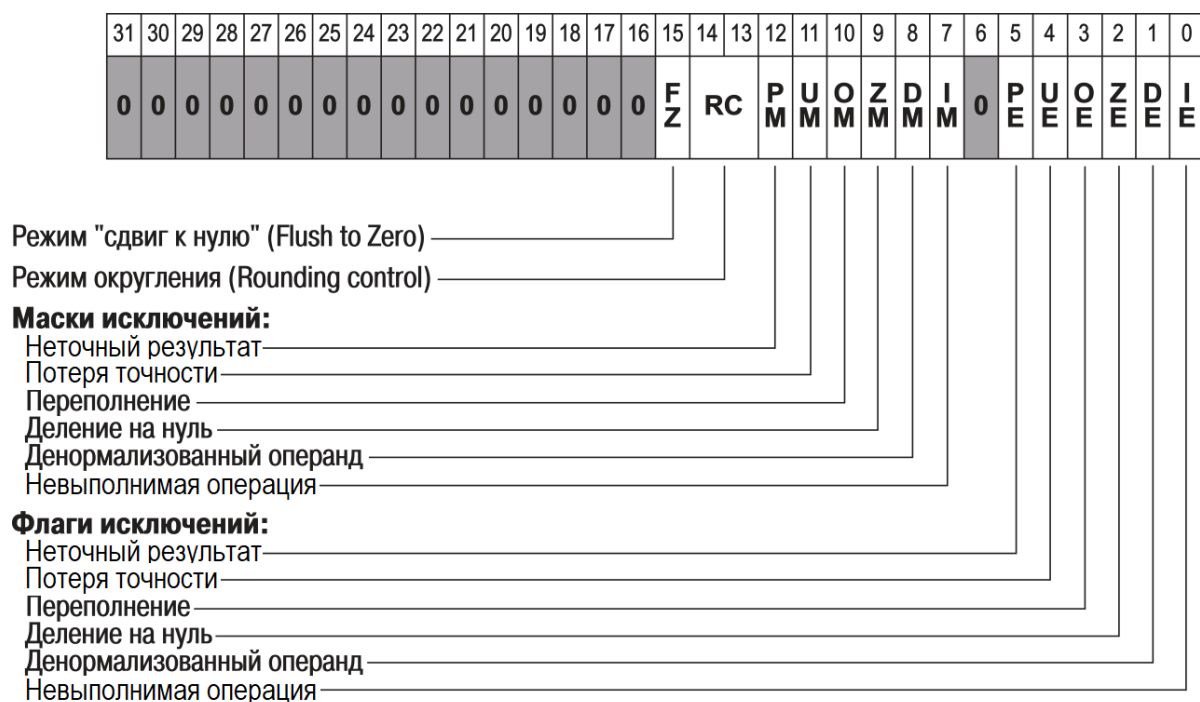


Рисунок 4.1 – Регистр MXCSR

Для сохранения регистра MXCSR в память используется инструкция STMXCSR. После сохранения регистра его копия в памяти может быть изменена, а затем загружена обратно инструкцией LDMXCSR. К примеру, маскировка всех исключений может производиться следующими инструкциями:

```

STMXCSR [addr]
OR word[addr], 0x1F80
LDMXCSR [addr]

```

3.6 Ограничения набора инструкций SSE

Набор SSE в базовом варианте не предоставляет инструкций для возведения в степень, логарифмирования и тригонометрических операций.

4 Задание на лабораторную работу

Каждый вариант содержит в себе 4 задания. Номер задания выбирается стандартным способом (номер в списке % количество заданий+1).

Задания 1 и 2 необходимо выполнить двумя способами:

- с помощью только инструкций сопроцессора x87;
- с помощью инструкций SSE (если в SSE нет инструкции с нужной математической операцией - использовать x87 для выполнения *только этой операции*).

Достаточно, чтобы программа работала для правильных значений параметров (например, вылет из-за деления на 0 или взятия логарифма отрицательного числа ошибкой не считается).

Организовывать ввод-вывод желательно, но не обязательно, за исключением задания 4, где требуется, как минимум, вывести 1(да) или 0(нет).

Для ввода-вывода float/double можно использовать макросы из предоставленного файла "io64_float.inc" (см. информацию в заголовке файла, разбирать весь файл необязательно).

Параметры a, b и c считать константами в пределах одного запуска (т.е. допустимо объявить их в секции .rodata).

В момент выхода из программы стек сопроцессора должен быть пуст.

Варианты заданий:

Легкий уровень:

Задание1: реализовать функцию, округляющую введенное пользователем вещественное число до ближайшего целого:

- 1) вверх
- 2) вниз

Задание 2. Вычислить:

1) $y = \frac{ax+b}{c}$	4) $y = \frac{(a-x)b}{c}$	7) $y = x^2 + \frac{a}{b}$
2) $y = \frac{a-bx}{c}$	5) $y = \frac{bc}{a+x}$	8) $y = \frac{x^2+a}{b}$
3) $y = \left(\frac{a}{x} + b\right)c$	6) $y = \frac{(x-a)c}{b}$	9) $y = \frac{x^2}{b} + a$

Задание 3. Решить уравнение:

1) $ax^2 + bx + c = 0$	2) $\frac{ax}{x+b} = c$	3) $\begin{cases} ax + b = 0, \\ cx = 1 \end{cases}$
------------------------	-------------------------	--

Задание 4. Проверить, что заданная точка (x, y) удовлетворяет условию:

1) $y < \log_2(x + a)$	3) $y > \ln x$	5) $y = \lg x$
2) $y \geq \sin(x/a)$	$y \leq \cos(ax)$	$y \neq \sin(x/a)$

Средний уровень:

Задание 1: см. легкий уровень

Задание 2. Вычислить:

1) $y = a \sin(bx) + \cos(cx)$	4) $y = e^{ax^2}$	7) $y = 2^{ax+b}$
2) $y = \operatorname{tg}(ax) + \operatorname{ctg}(bx)$	5) $y = \ln(x) + \log_2(x/a)$	8) $y = ax^b$
3) $y = \frac{\sin(x)}{ax} + \frac{b \cos(x)}{x}$	6) $y = \frac{\sin(x-a)}{c \cdot \cos b}$	9) $y = a[x] + b\{cx\}$

Задание 3. Решить уравнение:

1) $\cos(\ln(x + a)) = b$	2) $\log_2(\operatorname{tg}(x + a)) = b$	3) $2^{\operatorname{ctg}(x)} = a$
---------------------------	---	------------------------------------

Задание 4. Проверить, что заданная точка (x, y) удовлетворяет условию:

1) $y < \ln^3(\sin(x) + a)$	3) $y > \sinh x - a$	5) $y = \operatorname{th} a/x$
2) $y \geq \lg^2(\sin(x) + a)$	4) $y \leq \cosh(x - a)$	6) $y \neq \operatorname{cth}(x * a)$

Сложный уровень:

Задания 1, 3 и 4: то же, что и в среднем варианте.

Задание 2:

- 1) Реализовать приближенное вычисление косинуса инструкциями SSE.

- 2) Реализовать приближенное вычисление синуса инструкциями SSE.
- 3) Реализовать приближенное вычисление экспоненциальной функции инструкциями SSE.
- 4) Реализовать приближенное вычисление логарифма по основанию 2 инструкциями SSE в диапазоне $|x| < 1$.

Приложение А. Пример установки режима округления

```
%include "io64.inc"
section .bss
    result: resd 1

section .text
global main

set_round_up:
    sub rsp,8 ; allocate space on stack
    fstcw [rsp] ; save the control word
    mov al, [rsp+1] ; get the higher 8 bits
    and al, 0xF3 ; reset the RC field to 0
    or al, 8 ; set the RC field to 0x10
    mov [rsp+1], al
    fldcw [rsp] ; load the control word
    add rsp, 8 ; 'free' the allocated stack space
    ret

main:
    push rbp
    mov rbp, rsp
    ; call set_round_up
    fldl
    fldl
    fldl
    fadd
    fdiv ; the result is 0.5
    fist dword[result] ; round to integer
    mov eax, [result]
    PRINT_DEC 4, eax
    xor eax, eax
    leave
    ret
```

Приложение Б. Пример установки маски исключения

```
section .text
global main

enable_zd_exception:
    sub rsp, 8
    fstcw [rsp] ; store the control word
    mov al, [rsp] ; get the lower byte
    and al, 11 ; clear the bit 3 (ZDivision mask bit)
    mov [rsp] , al
    fldcw [rsp] ; load the control word
    add rsp, 8
    ret

main:
    push rbp
    mov rbp, rsp
    ; call enable_zd_exception
    fldl
    fldz
    fdiv ; 1/0
    fwait ; check and raise the exception
    leave
    xor eax, eax
    ret
```

Приложение В. Примеры решения задач

Задача 1: Вычислить $y = ax + bx$

ПОЛИЗ: $x \sin^2 a * b x * tg +$

Код для x87:

```
section .data
    a: dd 2.0
    b: dd 2.0
    x: dd 4.0
    y: dd 0.0

section .text
global CMAIN
CMAIN:
    fninit
    fld dword[x]
    fsin
    fmul st0
    fld dword[a]
    fmul
    fld dword[b]
    fld dword[x]
    fmul
    fptan
    fstp st0 ; pop 1.0 that fptan pushed
    fadd
    fstp dword[y]
    ret
```

Задача 2. Решить уравнение $\sqrt{1 - \frac{x}{a}} = b$

Замечание: нет решения при $b < 0$.

Решение: $x = a(1 - b^2)$.

ПОЛИЗ: $1 b b * - a *$

Код для x87:

```
section .data
    a: dd 2.0
    b: dd 2.0
    x: dd 4.0

section .text
global CMAIN
CMAIN:
    fninit
    fldl
    fld dword[b]
    fld dword[b]
    fmul
    fsub
    fld dword[a]
    fmul
    fstp dword[x]
    ret
```

Код для SSE:

```
section .data
    a: dd 2.0
    b: dd 2.0
    x: dd 0.0

section .text
global CMAIN
CMAIN:
    mov rbp, rsp; for correct debugging
    movss xmm0, [b]
    mov ecx, 1 ; we can't load constant into xmm
    cvtsi2ss xmm1, ecx ; so we convert it from ECX
    mulss xmm0, xmm0
    subss xmm1, xmm0
    mulss xmm1, [a]
    movss [x], xmm1
    ret
```

Задача 3. Проверить, что заданная точка (x, y) удовлетворяет условию $y < \ln(a - x)$.

Замечание: правая часть не определена на \mathbb{R} при $x > a$.

ПОЛИЗ правой части: $a x - \ln$

ПОЛИЗ логарифма: $\ln = \log_2 [\log_2 e]$ /, где $[\log_2 e]$ – константа.

Итоговая ПОЛИЗ: $a x - \log_2 [\log_2 e]$ /

Учитывая особенности FYL2X, ПОЛИЗ лучше привести к виду:

$1 [\log_2 e] / a x - (fyl2x)$

Код для x87:

```
%include "io64.inc"
section .data
    a: dd 4.0
    x: dd 2.0
    y: dd 1.5

section .text
global CMAIN
CMAIN:
    fninit
    fldl
    fldl2e
    fdiv ; 1/log2e
    fld dword[a]
    fld dword[x]
    fsub
    fyl2x
    fld dword[y]
    fcomip
    jnb false
    PRINT_DEC 4, 1
    jmp end
false:
    PRINT_DEC 4, 0
end:
    fstp st0 ; clear stack
    xor rax,rax
    ret
```