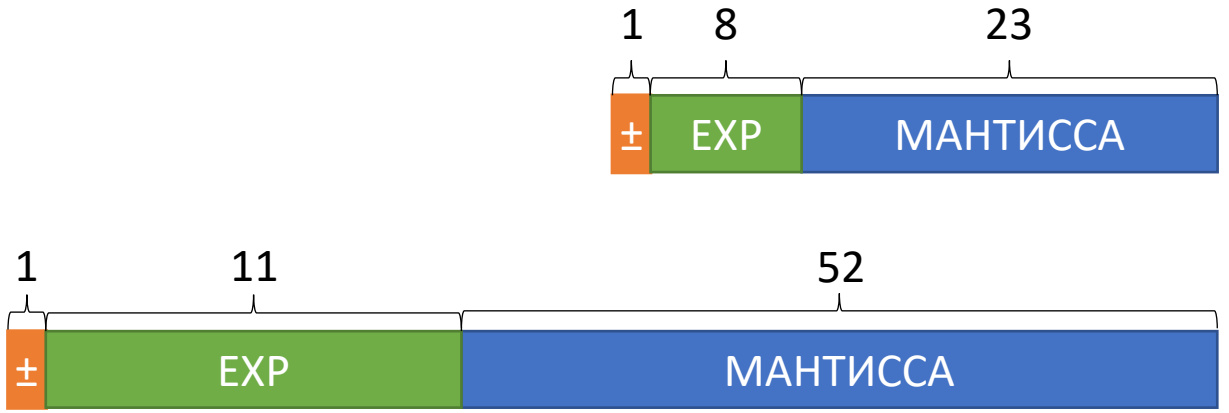
Низкоуровневое программирование

Лекция 2

Вычисления с плавающей запятой

Числа с плавающей запятой



Формат чисел с плавающей запятой определяется стандартом IEEE-754.

Первоначально, стандарт определял два основных типа чисел – числа одинарной точности и числа двойной точности размером 4 и 8 бит соответственно.

Помимо формата чисел, стандарт определяет также правила округления, сравнения, реакцию на запрещенные операции.

$$x = (-1)^s \cdot \mathbf{1}, m_1 m_2 m_3 \ldots m_{\mathrm{M}} \cdot 2^{exp-bias}$$

Точность числа	M	bias
Одинарная	23	127
Двойная	52	1023

Числа с плавающей запятой

Помимо самого порядка чисел, формат определяет также специальные значения: NaN, ± Infinity, ±0.

NaN (**N**ot **a** **N**umber) – это результат запрещенной операции: деления на 0, ситуации неопределенности (0/0, Inf/Inf, Inf*0) и пр. Данное число не равно никакому другому числу, а результат любой операции над NaN равен NaN.

Различают сигнальные NaN (**sNaN**), и “тихие” NaN (quiet NaN, **qNAN**). Возникновение сигнального NaN возбуждает исключение.

Тип числа	**Экспонента**	**Мантисса**
Нормализованное	$00\ldots01_2$ … $11\ldots10_2$	$00\ldots00_2$ ... $11\ldots11_2$
Денормализованное	$00\ldots00_2$	$00\ldots00_2$ ... $11\ldots11_2$
Бесконечность	$11\ldots11_2$	$00\ldots00_2$
sNaN	$11\ldots11_2$	$00\ldots01_2$ … $01\ldots11_2$
qNaN	$11\ldots11_2$	$10\ldots00_2$ … $11\ldots11_2$

Сопроцессор x87

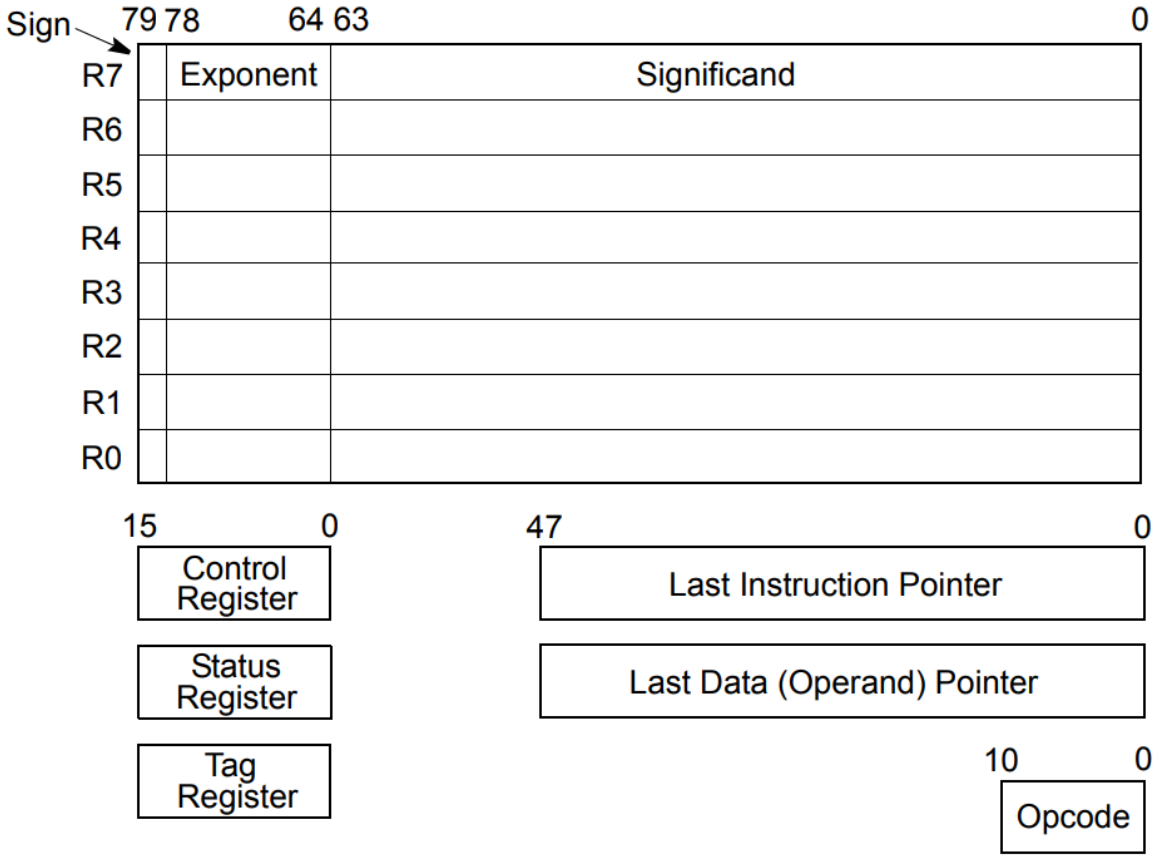
Процессор Intel 8086 не мог самостоятельно выполнять вычисления с плавающей запятой – для этого требовался отдельный сопроцессор (**FPU**, **f**loating **p**oint **u**nit). Выпускаемый Intel сопроцессор имел маркировку 8087. В отсутствие сопроцессора операции с плавающей точкой программно эмулировались (что влекло огромную потерю производительности).

FPU стал частью центрального процессора только в Intel 80486, выпущенном в 1989 г. – через 11 лет после Intel 8086.

Сопроцессор x87

Устройство сопроцессора практически не изменилось со времен 80387 – длина регистров увеличена не была, слово состояния и управляющее слово также почти не изменились.

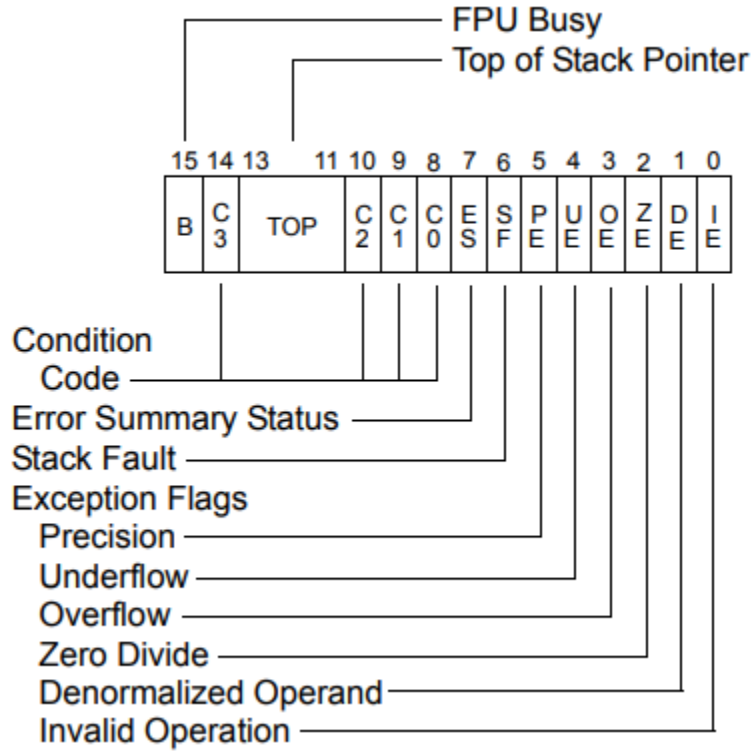
Набор регистров сопроцессора состоит из **8 регистров данных**, **регистра флагов**, **регистра состояния**, **регистра тэгов**, а также 3 служебных регистров, хранящих последний опкод, указатель на последние инструкцию и данные.



Регистр состояния

Регистр состояния содержит слово состояния сопроцессора. Отдельные биты слова являются флагами. Среди значимых флагов находятся **C0-C3**, которые хранят результаты сравнения, а в случае исключения – уточняющие данные; **TOP**, являющийся указателем на текущую вершину стека.

Остальные флаги являются флагами исключений.



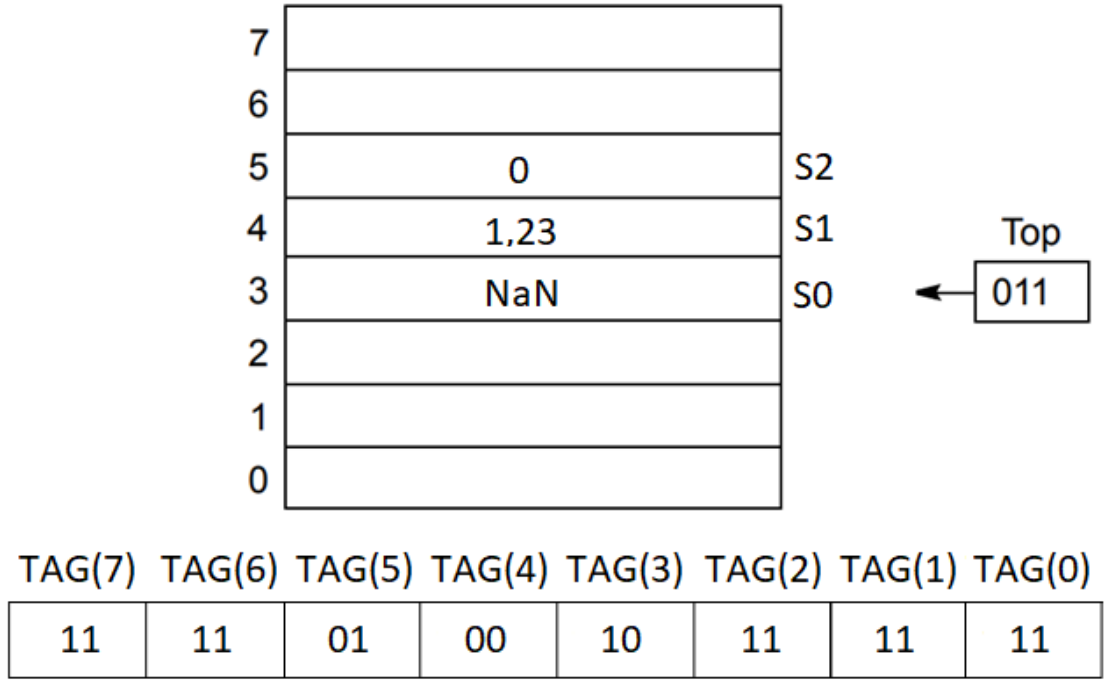
Стек FPU

Регистры данных не являются независимыми друг от друга – они организованы в **стек**. Номер регистра, являющегося текущей вершиной стека, хранится в поле TOP регистра состояния.

Состояние каждого регистра хранится в **регистре тегов**.

Загрузка данных и их **выгрузка** из стека производятся **с вершины стека**.

Адресация регистров осуществляется относительно текущей вершины: **ST(7)**, **ST(1), ST(0) = ST.**



Стек FPU растет вниз!

Инициализация и сохранение состояния сопроцессора

Для реинициализации состояния сопроцессора (сброса всех значений и состояния), используется инструкция `fninit`. По умолчанию после реинициализации все исключения маскированы (см. далее).

Сохранение полного состояния сопроцессора (108 байт) осуществляется инструкцией `fsave`.

Загрузка состояния осуществляется инструкцией `frstor`.

Типы данных FPU

По умолчанию, операции в FPU осуществляются с **расширенной точностью** – 80 бит вместо 64. Соответствующий тип в языке C – `long double`. Тем не менее, можно переключить FPU в режим вычислений с одинарной и двойной точности для строгого соответствия с IEEE-754.

Операции с целыми числами FPU не осуществляет (хотя мантисса занимает 64 бита – достаточно для точного представления long long). **Приведение типов** и расширение/уменьшение точности производятся **автоматически** при загрузке/выгрузке данных с FPU.

Загрузка вещественных чисел

Помещение данных в стек осуществляется инструкцией `fld`. Обычные регистры (RAX/EAX и др.) в качестве операнда указывать нельзя.

Тип загружаемого значения определяется операндом (4 байта – float, 8 байт – double, 10 байт – long double).

Для обмена данных между регистрами используется инструкция `fxch`.

`fld dword[rbx]`	TOP--; (стек растет вниз) ST0 = *(float*)RBX;
`fld qword[rbx]`	TOP--; ST0 = *(double*)RBX;
`fld tbyte[rbx]`	TOP--; ST0 = *(long double*)RBX;
`fld st5`	TOP-- ST0 = ST5;
`fxch st1, st2`	swap(ST1 , ST2)

Приведение целых чисел к вещественным

Загрузка целых чисел осуществляется инструкцией `fild`. Данная инструкция также принимает 1 операнд – адрес в памяти. Операнд автоматически приводится к числу расширенной точности.

Режим округления (вверх, вниз, до ближайшего, к 0) определяется кодом режима округления в слове состояния.

`fild dword[rbx]`	TOP--; ST0 = *(int*)RBX;
`fild qword[rbx]`	TOP--; ST0 = *(long long*)RBX;

Выгрузка данных

Выгрузка данных производится инструкциями `fst/fstp`
Инструкции имеют 1 операнд – адрес памяти, по которому производится сохранение, или один из регистров FPU.

Инструкция `fst` только сохраняет значение. Инструкция `fstp` (**F**PU **s**tore + **p**op) дополнительно удаляет знаение из вершины стека.

Целочисленные `fist/fistp` действуют аналогично.

`fst dword[rbx]`	*(float*)RBX=(float)ST0
`fstp tbyte[rbx]`	*(long double*)RBX=ST0 ; TOP++;
`fst st5`	ST5 = ST0;
`fstp st1`	ST1 = ST0; TOP++;
`fist dword[rbx]`	*(int*)RBX=(int)ST0 ;

Загрузка констант

Для загрузки констант используются специальные инструкции.

`FLD1`	TOP--; ST0 = 1.0;
`FLDL2T`	TOP--; ST0 = $\log_2 10$;
`FLDL2E`	TOP--; ST0 = $\log_2 e$;
`FLDPI`	TOP--; ST0 = π;
`FLDLG2`	TOP--; ST0 = $\log_{10} 2$;
`FLDLN2`	TOP--; ST0 = $\ln 2$;
`FLDZ`	TOP--; ST0 = $+0.0$;

Арифметические операции

Арифметические операции выполняются инструкциями `fadd/fsub/fsubr/fmul/fdiv/fdivr`. Данные инструкции принимают от 0 до 2 операндов. При этом, если операнды являются регистрами – то **один из них обязан быть ST0**.

Инструкции `faddp/fsubp/fsubrp/fmulp/fdivp/fdivrp` действуют аналогично обычным арифметическим операциям, однако они **дополнительно выталкивают значение с вершины стека**. Операндами могут быть только регистры сопроцессора.

Инструкции без операндов (например, `fadd` и `faddp)` являются синонимами в языке ассемблера NASM и всегда выталкивают значение с вершины.

Арифметические операции

`fadd`	ST1 += ST0; TOP++
`fsub tbyte[rbx]`	ST0 -= *(long double*)RBX
`fmul st5`	ST0 *= ST5;
`fdiv st1, st0`	ST1 /= ST0;
`fadd st0, st7`	ST0 += ST7
`fdiv st0, st1`	ST0 /=ST1
`fdivr st0, st1`	ST0 = ST1/ST0;
`faddp`	ST1 += ST0; TOP++;
`fmulp st5`	ST5 *= ST0; TOP++;
`fdivp st1, st0`	ST1 /= ST0; TOP++;
`fsub`	ST1=ST1-ST0; TOP++;
`fsubr`	ST1=ST0-ST1; TOP++;

Остаток от деления

Вещественный остаток от деления вычисляется инструкциями `fprem` и `fprem1` (Partial REMainder).

`fprem` функционирует, аналогично `std::remainder()`.

`fprem1` функционирует, аналогично `std::fmod()`.

Разница в значениях ST0 и ST1 не дожна превышать 2^{63}, иначе результатом инструкции является частичный остаток.

`fprem`	ST0 = ST0 % ST1
`fprem1`	ST0 = ST0 % ST1 (по IEEE-754)

Тригонометрические функции

Для вычисления тригонометрических функций используются инструкции

`fsin, fcos, fsincos,`

`fptan, fpatan`

`fsin`	ST0 = sin(ST0)
`fptan`	TOP--; ST1 = tan(ST0); **ST0 = 1**
`fpatan`	ST1 = arctan(ST1/ST0); TOP++;

`fsincos`	s = sin(ST0);
	c = cos(ST0);
	TOP--;
	ST0 = c; ST1 = s;

Квадратный корень и логарифм

Квадратный корень вычисляется инструкцией `fsqrt`.

`fsqrt`

ST0 = sqrt(ST0)

Двоичный логарифм вычисляется инструкцией `fyl2x`. Отметьте, что данная функция таже осуществляет умножение на второй элемент стека.

`fyl2x`

ST1 = ST1*$\log_2$ ST0;
TOP++;

Возведение в степень

Для возведения в дробную степень 2 используется инструкция `f2xm1`. Данная инструкция требует |ST0|<1.

Для возведения в целую степень 2 используется инструкция `fscale`. *Данная инструкция игнорирует дробную часть числа.*

Отделить целую и дробную части можно инструкцией `fperm`.

`f2xm1`	$ST0 = 2^{ST0} - 1, \lvert ST0 \rvert < 1$
`fscale`	$ST0 = ST0 * 2^{[ST1]}$

Сравнение

Для установки флагов регистра RFLAGS используются инструкции `fcomi/fcomip`. Инструкция `fcomip` после выполнения сравнения выталкивает значение из вершины стека.

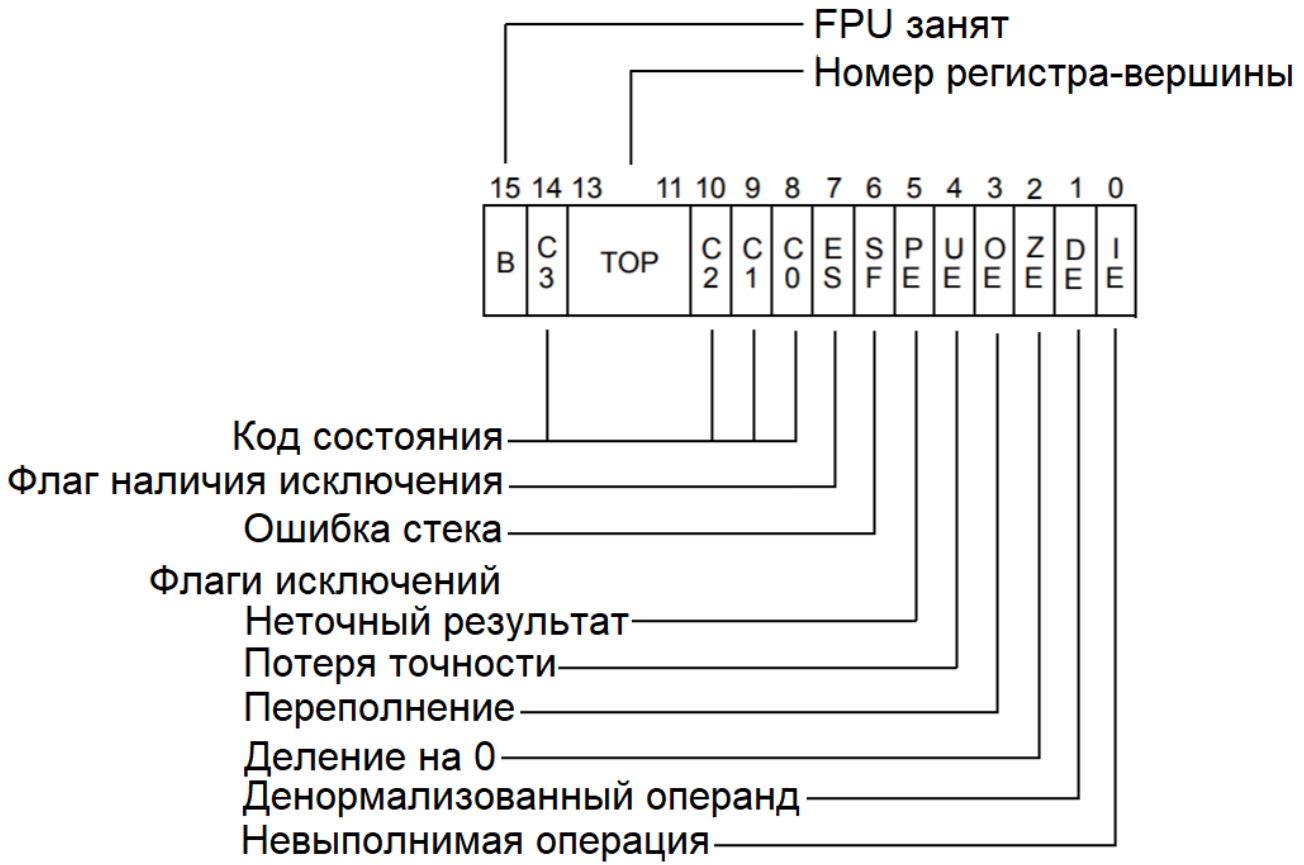
`fcomi`	compare(ST0, ST1)
`fcomip st3`	compare(ST0, ST3) TOP++

Результат сравнения	**ZF**	**PF**	**CF**
ST0 > ARG	0	0	0
ST0 < ARG	0	0	1
ST0 == ARG	1	0	0
ST0 is NaN || ARG is NaN	1	1	1

Исключения сопроцессора

В случае появления ошибки при вычислениях сопроцессор сигнализирует об этом процессору, который бросает *аппаратное* исключение.

Причину исключения можно узнать, анализируя соответствующие флаги регистра состояния.



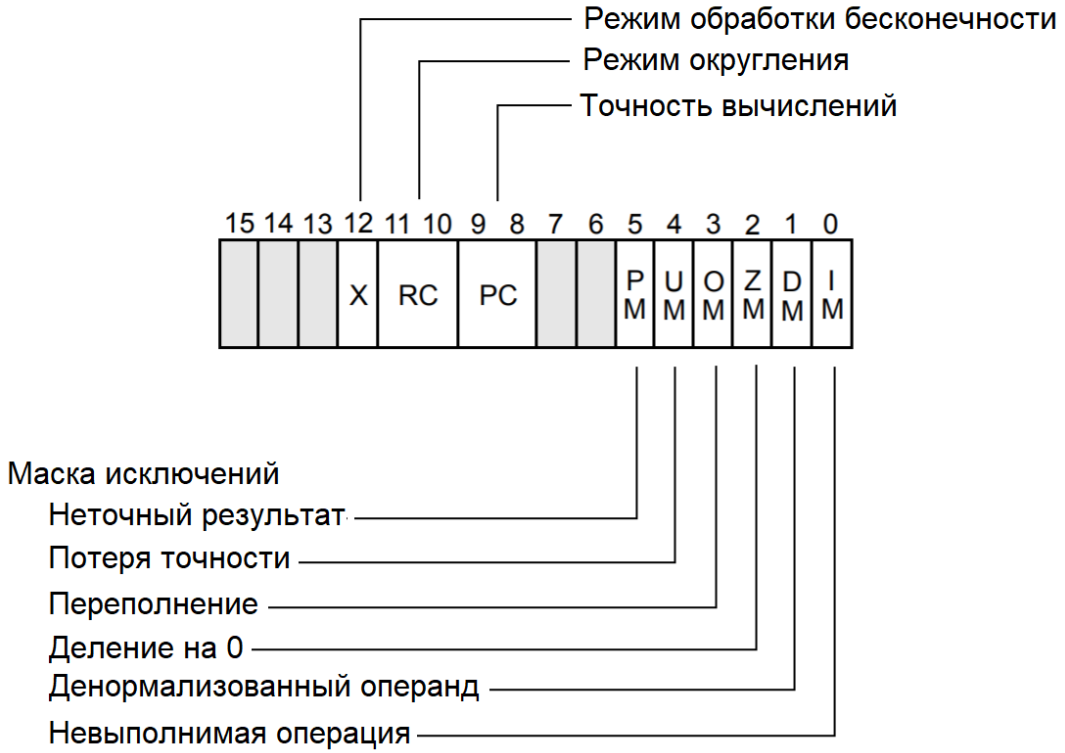
Управляющий регистр и маска исключений

(см. приложение Б ЛР2)

Исключения определенного типа могут быть явно запрещены (маскированы) путем установки флагов **управляющего регистра**.

В управляющем регистре можно также выбрать точность вычислений и метод округления.

Для того, чтобы изменить флаги, слово управления следует выгрузить в память инструкцией `fstcw`, изменить его, и загрузить обратно инструкцией `fldcw`.



Округление (см. приложение А ЛР2)

Поскольку в ходе вычислений с плавающей запятой неизбежно возникают автоматические округления в младшем разряде, а также явные округления до целого числа, в стандарте IEEE-754 определены 4 возможных режима округления.

Код режима указывается в поле RC управляющего регистра.

Режим округления	**Код**
К ближайшему целому	00_2
Вниз	01_2
Вверх	10_2
К нулю	11_2

Набор инструкций SSE

Хотя математический сопроцессор есть во всех современных x86-процессорах, вычисления с плавающей запятой используется более простой и современных подход.

В 1999 году был внедрен набор инструкций **SSE**. В настоящее время большинство процессоров x86 как минимум имеют набор SSE3.

SSE – это набор инструкций для векторных вычислений. Тем не менее, его можно использовать и для обычных вычислений с плавающей запятой.

В процессоре с поддержкой SSE3 есть 16 регистров **XMM0-XMM15**, которые независимы друг от друга (подобно RAX, RBX, R8 и др.).

Загрузка/выгрузка данных

Для загрузки данных в регистр и выгрузки данных из него используются инструкции `movss` (**mov**e **s**calar **s**ingle)и `movsd` (**mov**e **s**calar **d**ouble), загружающие в регистр число одинарной или двойной точности из памяти или другого регистра XMM.

Автоматического приведения типов в этом случае не происходит.

```
movss xmm0,[rbx]   XMM0 = *(float*)RBX

movsd [rbx],xmm0   *(double*)RBX = XMM0

movss xmm0,xmm1    XMM0 = XMM1
```

Приведение типов

Приведение между целочисленными 32-битными целыми числами и числами с плавающей запятой осуществляется инструкциями `cvtss2si/cvtsd2si` и `cvtsi2ss/cvtsi2sd`. Режим округления задается в регистре MXCSR.

Приемником для `cvtss2si/cvtsd2si` может быть только **регистр общего назначения**, а для `cvtsi2ss/cvtsi2sd` – только **XMM-регистр**

`cvtsi2ss xmm0,[rbx]`	XMM0 = *(float*)RBX
`cvtsi2sd xmm0,rax`	XMM0 = (double)RAX
`cvtss2si eax, xmm0`	EAX = (int)XMM0
`cvtss2si rax, xmm0`	RAX = (long long)XMM0

Приведение типов

Приведение float к double осуществляется инструкцией `cvtss2sd`. Приемником всегда должен быть XMM-регистр. Обратное преобразование осуществляется инструкцией `cvtsd2ss`.

`cvtss2sd xmm0,[rbx]`	XMM0 = (double)*(float*)RBX
`cvtss2sd xmm0,xmm1`	XMM0 = (double)XMM1
`cvtsd2ss xmm2,xmm1`	XMM2 = (float)XMM1

Математические операции

Математические операции выполняются инструкциями `addsX/subsX/divsX/mulsX/sqrtsX/rcpsX/rsqrtsX`, где `X` – `s` или `d`.

Смысл первых 5 инструкций очевиден из названия. Инструкция `rcpsX` вычисляет 1/arg. Инструкция `rsqrtsX` вычисляет 1/sqrt(arg).

```
addss xmm0,[rbx]        XMM0 += *(float*)RBX

subsd xmm1, xmm2        XXMM1 -= XMM2;
```

Сравнение (v1)

Сравнение чисел с плавающей точкой осуществляется инструкциями `cmp`**`eq`**`ss (==), cmp`**`lt`**`ss (<), cmp`**`le`**`ss (<=), cmp`**`unord`**`dss, cmp`**`neq`**`ss (!=), cmp`**`nlt`**`ss (>=), cmp`**`nle`**`ss (>), cmp`**`ord`**`ss.`

Результат сравнения записывается в регистр-приемник. Если сравнение вычисляется в **False,** приемник = **0**, если в **True**, то приемник= **0xFF..FF**.

`cmp`**`eq`**`ss XMM1, XMM2`	XMM1 = XMM1==XMM2 ? ~0 : 0
`cmp`**`nlt`**`ss XMM1, [rbx]`	XMM1 = XMM1>*RBX ? ~0 : 0

Сравнение (v2)

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.

```
comiss XMM1, XMM2

ucomiss XMM1, [rbx]
```

Результат сравнения	**ZF**	**PF**	**CF**
ARG1 > ARG2	0	0	0
ARG1 < ARG2	0	0	1
ARG1 == ARG2	1	0	0
ARG1 is NaN || ARG2 is NaN	1	1	1

Сравнение

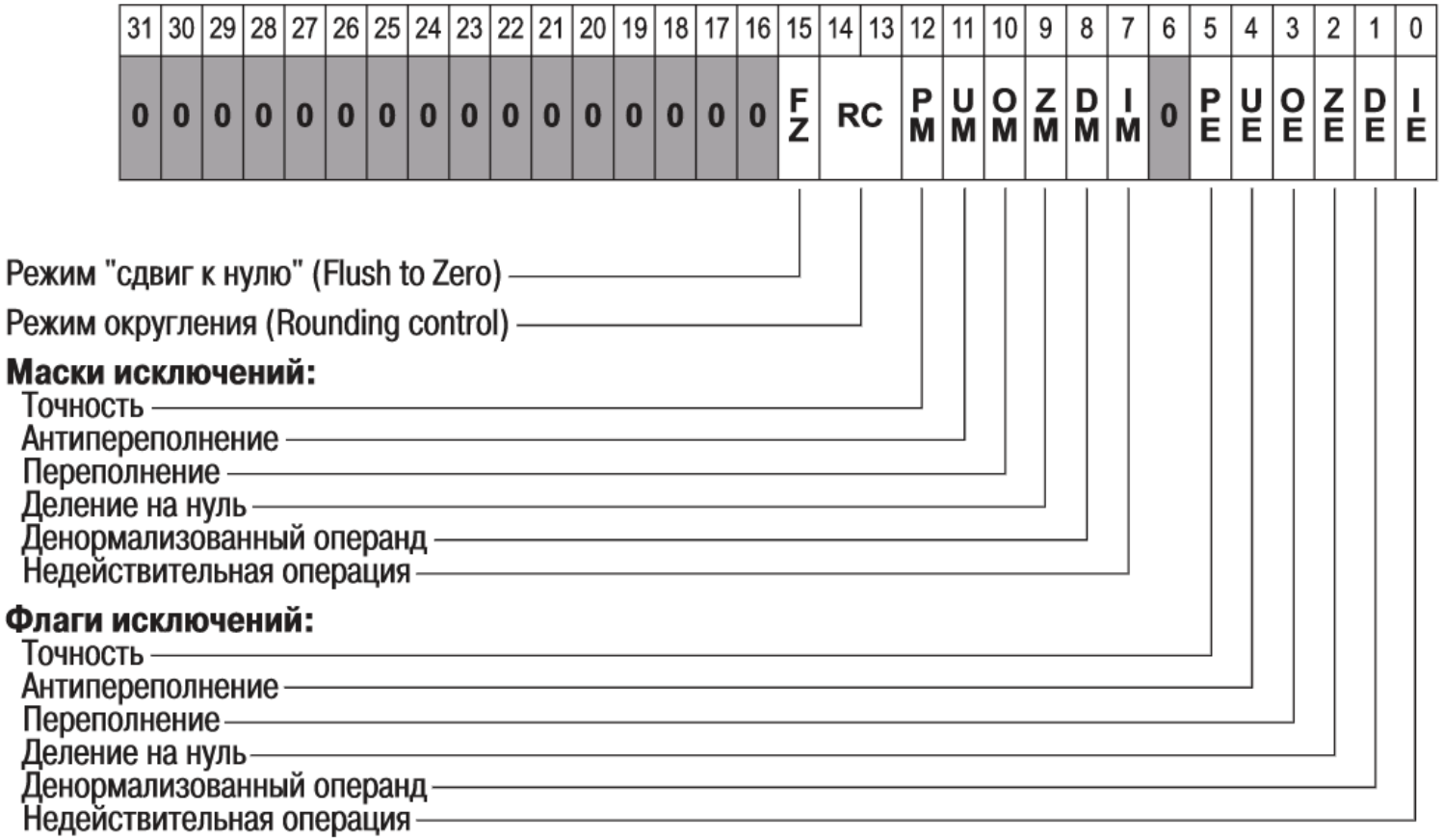
Иногда сравнение нужно только для определения наибольшего из 2 чисел. Вместо сравнения в комбинации с условным переходом, можно использовать инструкции `minsX/maxsX`.

```
minss XMM1, XMM2
```
XMM1 = XMM1<XMM2 ? XMM1 : XMM2

```
maxsd XMM1, [rbx]
```
XMM1 = XMM1>*RBX? XMM1 : *RBX

Регистр состояния и исключения

При возникновении ошибок во время вычислений, возникает аппаратное исключение. Тип исключения можно узнать, анализируя флаги регистра MXCSR. Здесь же можно отключить возникновение исключений и выбрать режим округления



Регистр состояния и исключения

Сохранение регистра MXCSR в память осуществляется инструкцией `stmxcsr`.

Загрузка содержимого регистра из памяти осуществляется инструкцией `ldmxcsr`.