МАТЕМАТИЧЕСКИЙ СОПРОЦЕССОР

Важной частью архитектуры микропроцессоров Intel является наличие устройства для обработки числовых данных в формате с плавающей точкой. До этого момента мы рассматривали команды и алгоритмы обработки целочисленных данных (чисел с фиксированной точкой).

Архитектура компьютеров на базе микропроцессоров вначале опиралась исключительно на целочисленную арифметику. С ростом мощи, а главное с осознанием разработчиками микропроцессорной техники того факта, что их устройства могут составить достойную конкуренцию своим «большим» предшественникам, в архитектуре компьютеров на базе микропроцессоров стали появляться устройства для обработки чисел с плавающей точкой. В архитектуре семейства микропроцессоров Intel 80x86 устройство для обработки чисел с плавающей точкой появилось в составе компьютера на базе микропроцессора i8086/88 и получило название математический сопроцессор (далее просто сопроцессор). Выбор такого названия был обусловлен тем, что, во-первых, это устройство было предназначено для расширения вычислительных возможностей основного процессора, а, во-вторых, оно было реализовано в виде отдельной микросхемы, то есть его присутствие было необязательным. Микросхема сопроцессора для микропроцессора i8086/88 имела название i8087. С появлением новых моделей микропроцессоров Intel совершенствовались и сопроцессоры, хотя их программная модель осталась практически неизменной. Как отдельные (а, соответственно, необязательные в конкретной комплектации компьютера) устройства, сопроцессоры сохранялись вплоть до модели микропроцессора i386 и имели название i287 и i387 соответственно. Начиная с модели i486, сопроцессор исполняется в одном корпусе с основным микропроцессором и, таким образом, является неотъемлемой частью компьютера.

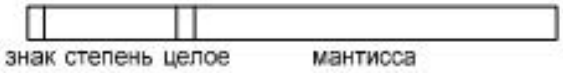
Перечислим основные возможности сопроцессора:

- полная поддержка стандартов IEEE-754 и 854 на арифметику с плавающей точкой. Эти стандарты описывают как форматы данных, с которыми должен работать сопроцессор, так и набор реализуемых им функций;

- поддержка численных алгоритмов для вычисления значений тригонометрических функций, логарифмов и т. п.;
- обработка десятичных чисел с точностью до 18 разрядов, что позволяет сопроцессору выполнять арифметические операции без округления над целыми десятичными числами со значениями до 1018;
- обработка вещественных чисел из диапазона $3.37x10^{-4932}...1.18x10^{+4932}$.

Представление чисел с плавающей точкой в разрядной сетке вычислительной машины

Числа с плавающей точкой представляются в следующей форме



Числа с плавающей точкой могут быть представлены в одном из трех форматов.

знаковое с плавающей точкой	±1.18e-38… 3.4e+38		float (DD)
знаковое с плавающей точкой двойной точности	±2.23e-308… 1.79e+308		double (DQ)
знаковое с плавающей точкой двойной расширенной точности	±3.37e-4932.. 1.18e+4932		long double (DT)

Числа простой и двойной точности (float и double, DD и DQ) могут быть представлены только в нормированной форме. При этом бит целой части числа является скрытым и «подразумевает» 1. Остальные 23 (52) разряда хранят двоичную мантиссу числа.

Числа двойной расширенной точности могут быть представлены как в нормированной, так и в ненормированной форме, поскольку бит целой части числа не является скрытым и может принимать значения как 0, так и 1.

Основным типом данных, которыми оперирует математический сопроцессор, являются 10-байтные данные (DT).

Ниже приведены способы кодировки чисел с плавающей точкой в формате DT.

Тип чисел	Знак	Степень	Целое	Мантисса
+∞	0	11…11	1	00…00
положительные нормированные	0	00…01 11…10	1	00…00 11…11
положительные ненормированные	0	00…00	0	00…00 11…11
ноль	0,1	00…00	0	00…00
отрицательные ненормированные	1	00…00	0	00…00 11…11
отрицательные нормированные	1	00…01 11…10	1	00…00 11…11
–∞	1	11…11	1	00…00
Нечисла NaN	x	11…11	1	xx…xx ≠0
float		8 бит	нет	23 бит
double		11 бит	нет	52 бит
long double		15 бит	1 бит	63 бит

Степень числа представляет собой степень числа 2 со сдвигом. Величина сдвига обусловлена необходимостью представления как чисел 1…∞, так и чисел 0…1 и зависит от типа представляемого числа и составляет соответственно

Тип данных	Сдвиг десятичный	Сдвиг двоичный
float	127	0111 1111
double	1023	011 1111 1111
long double	16383	011 1111 1111 1111

Пример1: Представить число 178,125 в 32-разрядной сетке.

Нормированное число в десятичной системе счисления $178,125 = 1,7825E_{10}2$.

Для представления числа в двоичной системе счисления преобразуем отдельно целую и дробную части:

$178_{10} = 10110010_2$.

$0,125_{10} = 0,001$.

Тогда $178,125_{10} = 10110010,001_2 = 1,0110010001E_2111$ (для преобразования в двоичную форму осуществляется сдвиг на 7 разрядов влево).

Для определения степени числа применяем сдвиг 0111111+00000111 = 10000110.

Таким образом, число $178,125_{10}$ запишется в разрядной сетке следующим образом:

зн.	степень								Мантисса																							
0	1	0	0	0	0	1	1	0	0	1	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Пример2: Представить число $12345678,1234567_{10}$ в 64-разрядной сетке.

Для представления числа в двоичной системе счисления преобразуем отдельно целую и дробную части:

$12345678_{10} = 1011\ 1100\ 0110\ 0001\ 0100\ 1110_2$.

$0{,}1234567_{10} = 0{,}0001\ 1111\ 1001\ 1010\ 1101\ 1011\ 1011\ 1_2$.

Тогда

$12345678{,}1234567_{10} = 101111000110000101001110{,}000111111001101011011011101111_2$

или $1{,}0111\ 1000\ 1100\ 0010\ 1001\ 1100\ 0011\ 1111\ 0011\ 0101\ 1011\ 0111\ 0111_2E_210111$ (сдвиг на 23 разрядов влево).

Для определения степени числа применяем сдвиг

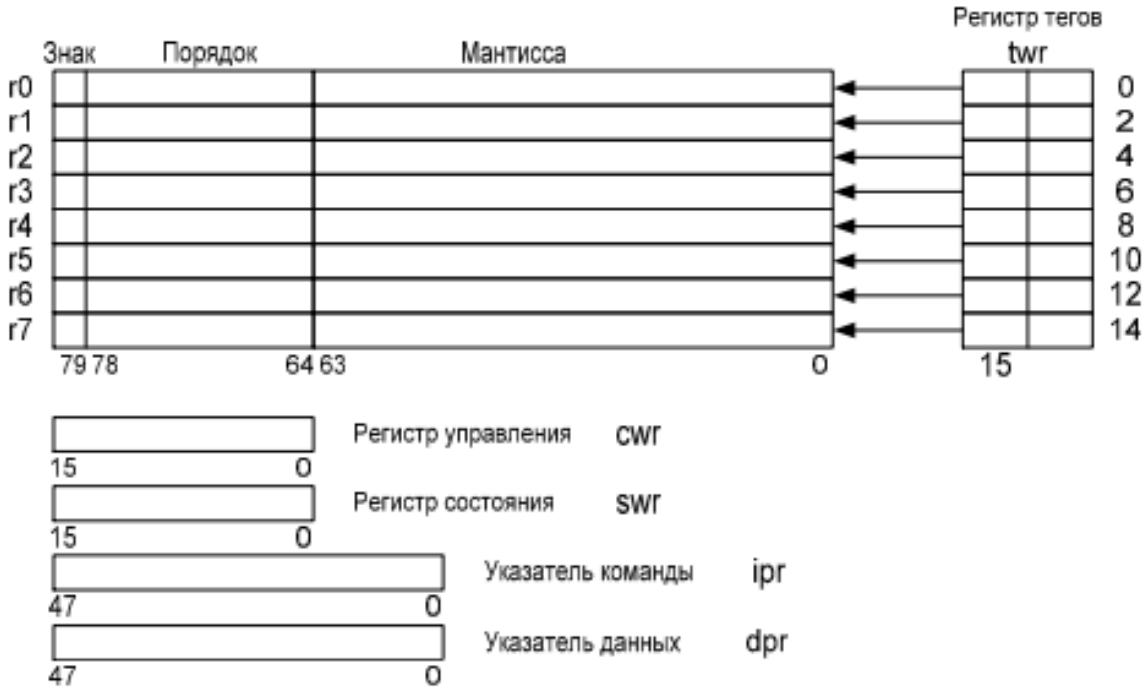
011 1111 1111 + 10111 = 10000010110.

Таким образом, число $12345678{,}1234567_{10}$ запишется в разрядной сетке следующим образом:

s	степень										мантисса																			
0	1	0	0	0	0	0	1	0	1	1	0	0	1	1	1	1	0	0	0	1	1	0	0	0	0	1	0	1	0	0	1
мантисса																														
1	1	0	0	0	0	1	1	1	1	1	1	0	0	1	1	0	1	0	1	1	0	1	1	0	1	1	1	0	1	1	1

Архитектура сопроцессора

Как и в случае с основным процессором, интерес для нас представляет программная модель сопроцессора. С точки зрения программиста, сопроцессор представляет собой совокупность регистров, каждый из которых имеет свое функциональное назначение.



В программной модели сопроцессора можно выделить три группы регистров.

1. Восемь регистров `r0...r7`, составляющих основу программной модели сопроцессора — стек сопроцессора. Размерность каждого регистра 80 битов. Такая организация характерна для устройств, специализирующихся на обработке вычислительных алгоритмов.
2. Три служебных регистра:

- регистр состояния сопроцессора `swr` (Status Word Register — регистр слова состояния) — отражает информацию о текущем состоянии сопроцессора. В регистре `swr` содержатся поля, позволяющие определить: какой регистр является текущей вершиной стека сопроцессора, какие исключения возникли после выполнения последней команды, каковы особенности выполнения последней команды (некий аналог регистра флагов основного процессора) и т. д.;
- управляющий регистр сопроцессора `cwr` (Control Word Register — регистр слова управления) — управляет режимами работы сопроцессора. С помощью полей в этом регистре можно регулировать точность выполнения численных вычислений, управлять округлением, маскировать исключения;

- регистр слова тегов `twr` (Tags Word Register — слово тегов) — используется для контроля за состоянием каждого из регистров `r0...r7`. Команды сопроцессора используют этот регистр, например, для того, чтобы определить возможность записи значений в эти регистры.

3. Два регистра указателей — данных `dpr` (Data Point Register) и команд `ipr` (Instruction Point Register). Они предназначены для запоминания информации об адресе команды, вызвавшей исключительную ситуацию и адресе ее операнда. Эти указатели используются при обработке исключительных ситуаций (но не для всех команд).

Все эти регистры являются программно доступными. Однако к одним из них доступ получить достаточно легко, для этого в системе команд сопроцессора существуют специальные команды. К другим регистрам получить доступ сложнее, так как специальных команд для этого нет, поэтому необходимо выполнить дополнительные действия.

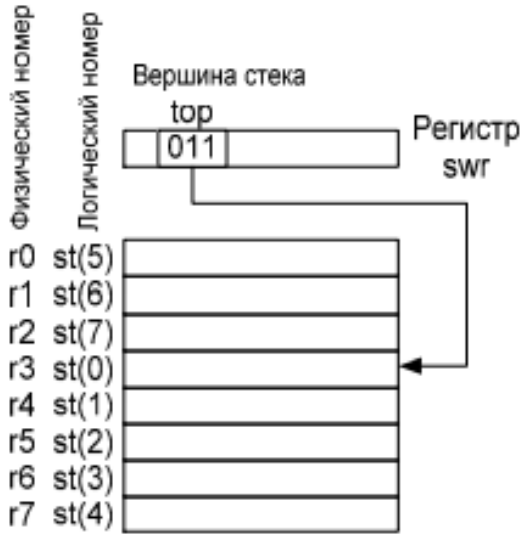
Рассмотрим общую логику работы сопроцессора и более подробно охарактеризуем перечисленные регистры.

Регистровый стек сопроцессора организован по принципу кольца. Это означает, что среди всех регистров, составляющих стек, нет такого, который является вершиной стека. Напротив, все регистры стека с функциональной точки зрения абсолютно одинаковы и равноправны. Но, как известно, в стеке всегда должна быть вершина. И она действительно есть, но является плавающей. Контроль текущей вершины осуществляется аппаратно с помощью трехбитового поля `top` регистра `swr`. В поле `top` фиксируется номер регистра стека 0...7 (`r0...r7`), являющегося в данный момент текущей вершиной стека.

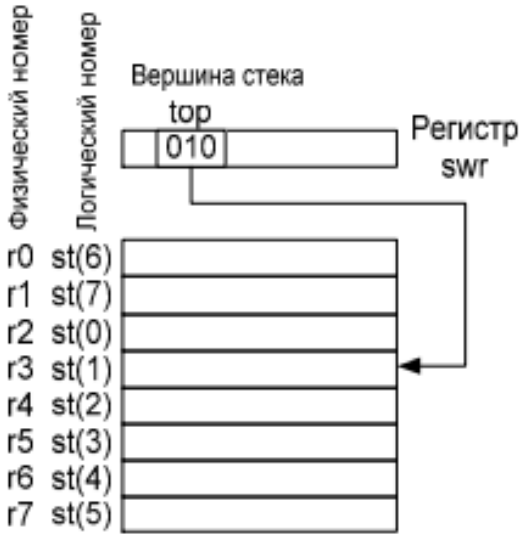
Команды сопроцессора не оперируют физическими номерами регистров стека `r0...r7`. Вместо этого они используют логические номера этих регистров `st(0)...st(7)`. С помощью логических номеров реализуется относительная адресация регистров стека сопроцессора. На рис. 9 показан пример, когда текущей вершиной до записи в стек является физический регистр стека `r3`, а после записи в стек текущей вершиной становится физический регистр стека `r2`. То есть, по мере записи в стек, указатель его вершины движется по направлению к младшим номерам

физических регистров (уменьшается на единицу). Если текущей вершиной является `r0`, то после записи очередного значения в стек сопроцессора его текущей вершиной станет физический регистр `r7`. Что касается логических номеров регистров стека `st(0)...st(7)`, то они «плавают» вместе с изменением текущей вершины стека. Таким образом, реализуется принцип кольца.

Такая организация стека обладает большой гибкостью. Это хорошо видно на примере передачи параметров подпрограмме. Для повышения гибкости разработки и использования подпрограмм не желательно привязывать их по передаваемым параметрам к аппаратным ресурсам (физическим номерам регистров сопроцессора). Гораздо удобнее задавать порядок следования передаваемых параметров в виде логических номеров регистров. Такой способ передачи был бы однозначным и не требовал от разработчика знания лишних подробностей об аппаратных реализациях сопроцессора. Логическая нумерация регистров сопроцессора, поддерживаемая на уровне системы команд, идеально реализует эту идею. При этом не имеет значения, в какой физический регистр стека сопроцессора были помещены данные перед вызовом подпрограммы, определяющим является только порядок следования параметров в стеке. По этой причине подпрограмме важно знать уже не место, а только порядок размещения передаваемых параметров в стеке.



до записи в стек



после записи в стек

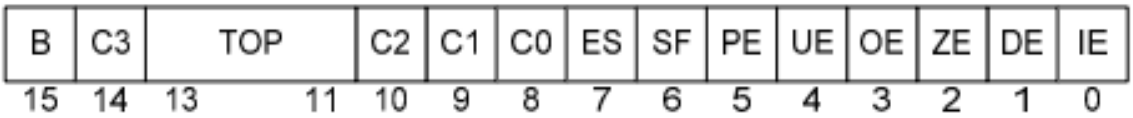
Процессор и сопроцессор имеют свои, несовместимые друг с другом системы команд и форматы обрабатываемых данных. Несмотря на то, что сопроцессор архитектурно представляет собой отдельное вычислительное устройство, он не может существовать отдельно от основного процессора.

Процессор и сопроцессор, являясь двумя самостоятельными вычислительными устройствами, могут работать параллельно. Но этот параллелизм касается только их внутренней работы над исполнением очередной команды. Оба процессора подключены к общей системной шине и имеют доступ к одинаковой информации. Инициирует процесс выборки очередной команды всегда основной процессор. После выборки команда попадает одновременно в оба процессора. Любая команда сопроцессора имеет код операции, первые пять бит, которого имеют значение 11011. Когда код операции начинается этими битами, то основной процессор по дальнейшему содержимому кода операции выясняет, требует ли данная команда обращения к памяти. Если это так, то основной процессор формирует физический адрес операнда и обращается к памяти, после чего содержимое ячейки памяти выставляется на шину данных. Если обращение к памяти не требуется, то основной процессор заканчивает работу над данной командой (не делая попытки ее исполнения!) и приступает к декодированию следующей команды из текущего входного командного потока. Что же касается сопроцессора, то выбранная команда попадает в него одновременно с основным процессором. Сопроцессор, определив по первым пяти битам, что очередная команда принадлежит его системе команд, начинает ее исполнение. Если команда требовала операнд из памяти, то сопроцессор обращается к шине данных за чтением содержимого ячейки памяти, которое к этому моменту предоставлено основным процессором. Из этой схемы взаимодействия следует, что в определенных случаях необходимо согласовывать работу обоих устройств. К примеру, если во входном потоке сразу за командой сопроцессора следует команда основного процессора, использующая результаты работы предыдущей команды, то сопроцессор не успеет выполнить свою команду за то время, когда основной процессор, пропустив сопроцессорную команду, выполнит свою. Очевидно, что логика работы программы будет нарушена. Возможна и другая ситуация. Если входной поток команд содержит последовательность из нескольких

команд сопроцессора, то процессор, в отличие от сопроцессора, проскочит их очень быстро, чего он не должен делать, так как он обеспечивает внешний интерфейс для сопроцессора. Эти и другие, более сложные ситуации, приводят к необходимости синхронизации между собой работы двух процессоров. В первых моделях микропроцессоров это делалось путем вставки перед или после каждой команды сопроцессора специальной команды `wait` или `fwait`. Работа данной команды заключалась в приостановке работы основного процессора до тех пор, пока сопроцессор не закончит работу над последней командой. В моделях микропроцессора (начиная с i486) подобная синхронизация выполняется командами `wait/fwait`, которые введены в алгоритм работы большинства команд сопроцессора. Но для некоторых команд из группы команд управления сопроцессором оставлена возможность выбора между командами с синхронизацией (ожиданием) и без нее.

Для организации эффективной работы с сопроцессором программист должен хорошо разобраться со структурой регистров и логикой их использования. Поэтому перед тем как приступить к рассмотрению команд и данных, с которыми работает сопроцессор, приведем описание структуры некоторых регистров сопроцессора.

Регистр состояния `swr` – отражает текущее состояние сопроцессора после выполнения последней команды.



Структурно регистр `swr` состоит из:

- 6 флагов исключительных ситуаций;
- бита `sf` (Stack Fault) — ошибка работы стека сопроцессора. Бит устанавливается в единицу, если возникает одна из трех исключительных ситуаций — `PE`, `UE` или `IE`. В частности, его установка информирует о попытке записи в заполненный стек, или, напротив, попытке чтения из пустого стека. После того как вы проанализировали этот бит, его нужно снова установить в ноль, вместе с битами `PE`, `UE` и `IE` (если они были установлены);

- бита `es` (Error Summary) — суммарная ошибка работы сопроцессора. Бит устанавливается в единицу, если возникает любая из шести перечисленных ниже исключительных ситуаций;
- четырех битов `c0…c3` (Condition Code) — кода условия. Назначение этих битов аналогично флагам в регистре `EFLAGS` основного процессора — отразить результат выполнения последней команды сопроцессора.
- трехбитного поля `top`. Поле содержит указатель регистра текущей вершины стека.

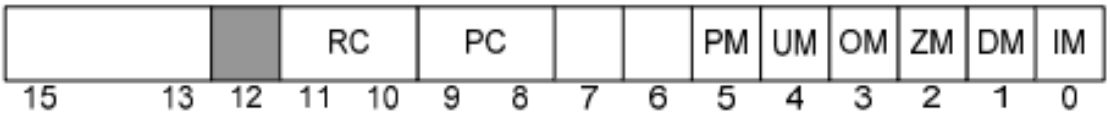
Почти половину регистра `swr` занимают биты (флаги) для регистрации исключительных ситуаций. **Исключения** — это разновидность прерываний, с помощью которых процессор информирует программу о некоторых особенностях ее реального исполнения. Сопроцессор также обладает способностью возбуждения подобных прерываний при возникновении определенных ситуаций (не обязательно ошибочных). Все возможные исключения сведены к шести типам, каждому из которых соответствует один бит в регистре `swr`. Программисту совсем не обязательно писать обработчик для реакции на ситуацию, приведшую к некоторому исключению. Сопроцессор умеет самостоятельно реагировать на многие из них. Это так называемая обработка исключений по умолчанию. Для того чтобы «заказать» сопроцессору обработку определенного типа исключения по умолчанию, необходимо это исключение замаскировать. Такое действие выполняется с помощью установки в единицу нужного бита в управляющем регистре сопроцессора `cwr` (рис. 10). Приведем типы исключений, фиксируемые с помощью регистра `swr`:

- `IE` (Invalide operation Error) — недействительный код операция;
- `DE` (Denormalized operand Error) — ненормированный операнд;
- `ZE` (divide by Zero Error) — ошибка деления на нуль;
- `OE` (Overflow Error) — ошибка переполнения. Возникает в случае выхода порядка числа за максимально допустимый диапазон;
- `UE` (Underflow Error) — ошибка антипереполнения. Возникает, когда результат слишком мал;

- PE (Precision Error) — ошибка точности. Устанавливается, когда сопроцессору приходится округлять результат из-за того, что его точное представление невозможно. Так, сопроцессору никогда не удастся точно разделить 10 на 3.

При возникновении любого из этих шести типов исключений устанавливается в единицу соответствующий бит в регистре swr, вне зависимости от того, было ли замаскировано это исключение в регистре cwr или нет.

Регистр управления работой сопроцессора cwr – определяет особенности обработки числовых данных.



Он состоит из:

- шести масок исключений;
- поля управления точностью PC (Precision Control);
- поля управления округлением RC (Rounding Control).

Шесть масок предназначены для маскирования исключительных ситуаций, возникновение которых фиксируется с помощью шести бит регистра swr. Если какие-то маскирующие биты исключений в регистре cwr установлены в единицу, то это означает, что соответствующие исключения будут обрабатываться самим сопроцессором. Если для какого-либо исключения в соответствующем бите масок исключений регистра cwr содержится нулевое значение, то при возникновении исключения этого типа будет возбуждено прерывание 16 (10h). Операционная система должна содержать (или программист должен написать) обработчик этого прерывания. Он должен выяснить причину прерывания, после чего, если это необходимо, исправить ее, а также выполнить другие действия.

Поле управления точностью PC предназначено для выбора длины мантиссы. Возможные значения в этом поле означают:

- PC=00 — длина мантиссы 24 бита;
- PC=10 — длина мантиссы 53 бита;
- PC=11 — длина мантиссы 64 бита.

По умолчанию устанавливается значение поля PC=11.

Поле управления округлением RC позволяет управлять процессом округления чисел в процессе работы сопроцессора. Необходимость операции округления может появиться в ситуации, когда после выполнения очередной команды сопроцессора получается не представимый результат, например, периодическая дробь 3,333…. Установив одно из значений в поле RC, можно выполнить округление в необходимую сторону.

Значения поля RC с соответствующим алгоритмом округления:

- 00 — значение округляется к ближайшему числу, которое можно представить в разрядной сетке регистра сопроцессора;
- 01 — значение округляется в меньшую сторону;
- 10 — значение округляется в большую сторону;
- 11 — производится отбрасывание дробной части числа. Используется для приведения значения к форме, которая может использоваться в операциях целочисленной арифметики.

Регистр тегов twr – представляет собой совокупность двухбитовых полей. Каждое поле соответствует определенному физическому регистру стека и характеризует его текущее состояние. Изменение состояния любого регистра стека отражается на содержимом соответствующего этому регистру поля регистра тега. Возможны следующие значения в полях регистра тега:

- 00 — регистр стека сопроцессора занят допустимым ненулевым значением;
- 01 — регистр стека сопроцессора содержит нулевое значение;
- 10 — регистр стека сопроцессора содержит одно из специальных численных значений, за исключением нуля;
- 11 — регистр пуст и в него можно производить запись. Нужно отметить, что это значение в одном из двухбитовых полей регистра тегов не означает, что все биты соответствующего регистра стека должны быть обязательно нулевыми.

Поскольку при написании программы разработчик манипулирует не абсолютными, а относительными номерами регистров стека, у него могут возникнуть трудности при попытке интерпретации содержимого регистра тегов twr, с соответствующими физическими регистрами стека. В качестве связующего звена необходимо привлекать информацию из поля top регистра swr.

Система команд сопроцессора

Система команд сопроцессора включает в себя около 80 машинных команд, включающих в себя

- команды передачи данных;
- команды сравнения данных;
- арифметические команды;
- трансцендентные команды;
- команды управления.

Мнемоническое обозначение команд сопроцессора характеризует особенности их работы и в связи с этим может представлять определенный интерес. Поэтому коротко рассмотрим основные моменты образования названий команд:

- все мнемонические обозначения начинаются с символа `f` (`float`);
- вторая буква мнемонического обозначения определяет тип операнда в памяти, с которым работает команда:

  `i` — целое двоичное число;
  `b` — целое десятичное число;
  отсутствие буквы — вещественное число;

- последняя буква мнемонического обозначения команды `p` означает, что последним действием команды обязательно является извлечение операнда из стека;
- последняя или предпоследняя буква `r` (reversed) означает реверсивное следование операндов при выполнении команд вычитания и деления, так как для них важен порядок следования операндов.

Система команд сопроцессора отличается большой гибкостью в выборе вариантов задания команд, реализующих определенную операцию, и их операндов. Минимальная длина команды сопроцессора — 2 байта.

Все команды сопроцессора оперируют регистрами стека сопроцессора. Если операнд в команде не указывается, то по умолчанию используется вершина стека сопроцессора (логический регистр st(0)). Если команда выполняет действие с двумя операндами по умолчанию, то эти операнды – регистры st(0) и st(1).

Команда	Операнды	Флаги				Пояснение	Описание
		C3	C2	C1	C0		
Команды передачи данных							
Вещественного типа							
FLD	src	U	U	+	U	TOP--1;ST(0)=src;	Загрузка в стек
FST	dst	U	U	+	U	dst=ST(0);	Пересылка в память
FSTP	dst	U	U	+	U	dst=ST(0);TOP+=1;	Пересылка в память с выталкиванием из стека
FXCH	[ST(i)]	U	U	+	U	ST(0)↔ST(i)	Обмен значений ST(0) и ST(i)
Целого типа							
FILD	src	U	U	+	U	TOP--1;ST(0)=src;	Загрузка в стек
FIST	src	U	U	+	U	dst=ST(0);	Пересылка в память
FISTP	dst	U	U	+	U	dst=ST(0);TOP+=1;	Пересылка в память с выталкиванием из стека
Двоично-десятичного типа							
FBLD	src	U	U	+	U	TOP--1;ST(0)=src;	Загрузка в стек
FBSTP	dst	U	U	+	U	dst=ST(0);TOP+=1;	Пересылка в память с выталкиванием из стека
Команды загрузки констант							
FLDZ		U	U	+	U	TOP--1;ST(0)=0;	Загрузка 0
FLD1		U	U	+	U	TOP--1;ST(0)=1;	Загрузка 1
FLDPI		U	U	+	U	TOP--1;ST(0)=3.1415926535;	Загрузка π
FLDL2T		U	U	+	U	TOP--1;ST(0)=3,3219280948;	Загрузка $\log_2 10$
FLDL2E		U	U	+	U	TOP--1;ST(0)=1,4426950408;	Загрузка $\log_2 e$
FLDLG2		U	U	+	U	TOP--1;ST(0)=0,3010299956;	Загрузка lg 2
FLDLN2		U	U	+	U	TOP--1;ST(0)=0,6931471805;	Загрузка ln 2
Команды сравнения данных							
Вещественного типа							
FCOM FUCOM	[src]	+	+	+	+	ST(0)-src;	Сравнение вещественное
FCOMP FUCOMP	[src]	+	+	+	+	ST(0)-src; TOP+=1;	Сравнение вещественное с выталкиванием
FCOMPP FUCOMPP		+	+	+	+	ST(0)-ST(1); TOP+=2;	Сравнение вещественное с выталкиванием
FCOMI FUCOMI	ST,[ST(i)]	-	-	-	+	ST(0)-ST(i);	Сравнение вещественное с модификацией EFLAGS
FCOMIP FUCOMIP	ST,[ST(i)]	-	-	-	+	ST(0)-ST(i); TOP+=1;	Сравнение вещественное с выталкиванием и модификацией EFLAGS
FXAM		+	+	+	+		Анализ ST(0)
Целого типа							
FICOM	src	+	+	+	+	ST(0)-src;	Сравнение целочисленное
FICOMP	src	+	+	+	+	ST(0)-src; TOP+=1;	Сравнение целочисленное с выталкиванием
FTST		+	+	+	+	ST(0)-0.0;	Сравнение с нулем
Арифметические команды							
Целого типа							
FIADD	src	U	U	+	U	ST(0)=ST(0)+src;	Сложение целочисленное
FISUB	src	U	U	+	U	ST(0)=ST(0)-src;	Вычитание целочисленное
FISUBR	src	U	U	+	U	ST(0)=src-ST(0);	Вычитание целочисленное реверсивное
FIMUL	src	U	U	+	U	ST(0)= ST(0)*src;	Умножение целочисленное
FIDIV	src	U	U	+	U	ST(0)= ST(0)/src;	Деление целочисленное
FIDIVR	src	U	U	+	U	ST(0)= src/ST(0);	Деление целочисленное реверсивное

Команда	Операнды	Флаги				Пояснение	Описание
		C3	C2	C1	C0		
Вещественного типа							
FADD	[dst],src	U	U	+	U	dst=dst+src;	Сложение вещественное
FADDP	[ST(i), ST(0)]	U	U	+	U	ST(i)=ST(i)+ST(0);TOP+=1;	Сложение вещественное с выталкиванием
FSUB	[dst],src	U	U	+	U	dst=dst-src;	Вычитание вещественное
FSUBR	[dst],src	U	U	+	U	dst=src-dst;	Вычитание вещественное реверсивное
FSUBP	[ST(i), ST(0)]	U	U	+	U	ST(i)=ST(i)-ST(0);TOP+=1;	Вычитание вещественное с выталкиванием
FSUBRP	[ST(i), ST(0)]	U	U	+	U	ST(i)=ST(0)-ST(i);TOP+=1;	Вычитание вещественное реверсивное с выталкиванием
FMUL	[dst],src	U	U	+	U	dst=dst*src;	Умножение вещественное
FMULP	[ST(i), ST(0)]	U	U	+	U	ST(i)=ST(i)*ST(0);TOP+=1;	Умножение вещественное с выталкиванием
FDIV	[dst],src	U	U	+	U	dst=dst/src;	Деление вещественное
FDIVR	[dst],src	U	U	+	U	dst=src/dst;	Деление вещественное реверсивное
FDIVP	[ST(i), ST(0)]	U	U	+	U	ST(i)=ST(i)/ST(0);TOP+=1;	Деление вещественное с выталкиванием
FDIVRP	[ST(i), ST(0)]	U	U	+	U	ST(i)=ST(0)/ST(i);TOP+=1;	Деление вещественное реверсивное с выталкиванием
Дополнительные арифметические команды							
FSQRT		U	U	+	U	$ST(0) = \sqrt{ST(0)}$	Вычисление квадратного корня
FABS		U	U	+	U	$ST(0) = \lvert ST(0) \rvert$	Вычисление модуля
FCHS		U	U	+	U	ST(0)=-ST(0)	Изменение знака
FXTRACT		U	U	+	U	temp = ST(0); ST(0)=порядок(temp); TOP--=1; ST(0)=мантисса(temp);	Выделение порядка и мантиссы
FSCALE		U	U	+	U	$ST(0) = ST(0) * 2^{ST(1)}$	Масштабирование по степеням 2
FRNDINT		U	U	+	U	ST(0)=(ST(0))	Округление ST(0)
FPREM		+	+	+	+	ST(0)=ST(0)-Q*ST(1);	Частичный остаток от деления
FPREM1		+	+	+	+	ST(0)=ST(0)-Q*ST(1), ST(0)<-ST(1)/2	Частичный остаток от деления
Команды трансцендентных функций							
FCOS		U	+	+	U	ST(0)=cos(ST(0));	Вычисление косинуса
FSIN		U	+	+	U	ST(0)=sin(ST(0));	Вычисление синуса
FSINCOS		U	+	+	U	temp=ST(0), ST(0)=sin(temp); TOP--=1;ST(0)=cos(temp);	Вычисление синуса и косинуса
FPTAN		U	+	+	U	ST(0)=tg(ST(0)); TOP--=1; ST(0)=1.0;	Вычисление тангенса
FPATAN		U	U	+	U	ST(1)=atan(ST(1)/ST(0)); TOP+=1;	Вычисление арктангенса
F2XM1		U	U	+	U	$ST(0)=2^{ST(0)}-1;$	Вычисление выражения $y = 2^x - 1$
FYL2X		U	U	+	U	x=ST(0);y=ST(1);TOP+=1; $ST(0)=y*\log_2 x;$	Вычисление выражения $z = y \cdot \log_2 x$
FYL2XP1		U	U	+	U	x=ST(0);y=ST(1);TOP+=1; $ST(0)=y*\log_2 x;$	Вычисление выражения $z = y \cdot \log_2 x$

Команда	Операнды	Флаги				Пояснение	Описание
		C3	C2	C1	C0		
Команды управления сопроцессором							
FWAIT		-	-	-	-		Синхронизация работы с основным процессором
FINIT FNINIT		-	-	-	-	CWR=037Fh; SWR=0; TWR=FFFFh; DPR=0; IPR=0;	Инициализация сопроцессора
FSTSW FSTNSW	dst AX	-	-	-	-	dst=SWR; AX = SWR;	Считать слово состояния сопроцессора в память
FSTCW FSTNCW	Dst AX	U	U	U	U	dst=CWR; AX = CWR;	Считать слово управления сопроцессора в память
FLDCW	src	U	U	U	U	CWR=src;	Загрузить слово управления сопроцессора
FCLEX FNCLEX		U	U	U	U	SWR=SWR & 7F00h	Сброс флагов исключений
FINCSTP		U	U	U	U	TOP+=1;	Увеличение указателя стека сопроцессора на 1
FDECSTP		U	U	U	U	TOP-=1;	Уменьшение указателя стека сопроцессора на 1
FFREE	ST(i)	U	U	U	U	TAG(i)=11b	Очистка указанного регистра
FNOP		-	-	-	-		Пустая операция
FSAVE FNSAVE	dst	0	0	0	0	...	Сохранение состояния среды сопроцессора
FRSTOR	src	+	+	+	+	...	Восстановление состояния среды сопроцессора
FSTENV FNSTENV	dst	U	U	U	U	...	Частичное сохранение состояния среды сопроцессора
FLDENV	src	+	+	+	+	...	Восстановление частичного состояния среды сопроцессора

Команды передачи данных

Группа команд передачи данных предназначена для организации обмена между регистрами стека, вершиной стека сопроцессора и ячейками оперативной памяти. Команды этой группы имеют такое же значение для процесса программирования сопроцессора, как и команда `mov` основного процессора. С помощью этих команд осуществляются все перемещения значений операндов в сопроцессор и из него. По этой причине для каждого из трех типов данных, с которыми может работать сопроцессор, существует своя подгруппа команд передачи данных. Собственно на этом уровне все его умения по работе с различными форматами данных и заканчиваются. Главной функцией всех команд загрузки данных в сопроцессор является преобразование их к единому представлению в виде вещественного числа расширенного формата. Это же касается и обратной операции — сохранения в памяти данных из сопроцессора.

Команды передачи данных можно разделить на следующие группы:

- команды передачи данных в вещественном формате;
- команды передачи данных в целочисленном формате;
- команды передачи данных в двоично-десятичном формате.

Команды передачи данных в вещественном формате:

Команда `FLD` – загрузка вещественного числа из области памяти в вершину стека сопроцессора.

Синтаксис: `FLD источник`

Помещает операнд `источник` в вершину стека сопроцессора. Операнд может быть простой, двойной или двойной расширенной точности. Если операнд задан в формате простой или двойной точности, он автоматически преобразуется к форматы двойной расширенной точности перед помещением в стек.

Устанавливает в 1 признак `C1` при переполнении стека.

Команда `FST/FSTP` – сохранение вещественного числа из вершины стека сопроцессора в память.

Синтаксис: `FST/FSTP приемник`

Копирует значение из вершины стека сопроцессора в операнд `приемник`. Операнд может быть ячейкой памяти или другим регистром стека сопроцессора. При сохранении значения в памяти оно автоматически преобразуется в формат с простой или двойной точностью. Если операндом является ячейка памяти, то она определяет адрес первого байта операнда-приемника. Если операндом является регистр, то он определяет регистр в стеке сопроцессора относительно вершины стека, определенной полем `top` регистра `swr`.

Признак `C1` определяет направление округления.

`C1=1` – округление вверх, `C1=0` – округление вниз.

Команда `FSTP` –в отличие от `FST` осуществляет выталкивание вещественного числа из стека после его сохранения в памяти. Команда изменяет поле `top`, увеличивая его на единицу. Вследствие этого, вершиной стека становится следующий, больший по своему физическому номеру, регистр стека сопроцессора.

Команды передачи данных в целочисленном формате:

Команда `FILD` – загрузка целого числа из памяти в вершину стека сопроцессора.

Синтаксис: `FILD источник`

Помещает знаковый целочисленный операнд `источник` в вершину стека сопроцессора. Операнд может быть словом, двойным словом или 8-байтным словом. Значение операнда перед помещением в стек преобразуется к формату с плавающей точкой двойной расширенной точности.

Устанавливает в 1 признак `C1` при переполнении стека.

Команда `FIST/FISTP` – сохранение целого числа из вершины стека сопроцессора в память.

Синтаксис: `FIST/FISTP приемник`

Преобразует значение из вершины стека сопроцессора в знаковое целое (в соответствии с таблицей) и сохраняет его в операнде `приемник`. Значение может быть сохранено в целочисленном формате слова или двойного слова. Операнд определяет адрес первого байта сохраняемого значения.

ST(0)	DEST
-∞ или очень маленькое число	недопустимая операция #IA

ST(0) ≤ -1	int со знаком «-»
-1< ST(0) < -0	0 или -1 в зависимости от режима округления
-0, +0	0
+0< ST(0) < +1	0 или +1 в зависимости от режима округления
ST(0) ≥ +1	int со знаком «+»
+∞ или очень большое число	недопустимая операция #IA
нечисло (NaN)	недопустимая операция #IA

Признак C1 определяет направление округления.

C1=1 – округление вверх, C1=0 – округление вниз.

Команды передачи данных в десятичном формате:

Команда FBLD – загрузка десятичного числа из памяти в вершину стека сопроцессора.

Синтаксис: FBLD источник

Помещает знаковый двоично-десятичный операнд источник в вершину стека сопроцессора, преобразуя его в формат с плавающей точкой двойной расширенной точности. Знак операнда сохраняется за исключением значения -0.

Устанавливает в 1 признак C1 при переполнении стека.

Команда FBSTP – сохранение десятичного числа из вершины стека сопроцессора в области памяти.

Синтаксис: FBSTP приемник

Значение выталкивается из стека после преобразования его в формат двоично-десятичного упакованного числа и сохраняется в операнде приемник. Для двоично-десятичных чисел нет команды сохранения значения в памяти без выталкивания из стека.

Признак C1 определяет направление округления.

C1=1 – округление вверх, C1=0 – округление вниз.

К группе команд передачи данных можно отнести команду обмена вершины регистрового стека st(0) с любым другим регистром стека.

Команда FXCH – обмен значений между текущей вершиной стека и регистром стека сопроцессора st(i).

Синтаксис: FXCH [операнд]

Производит обмен значений вершины стека ST(0) и стекового регистра ST(i). Если операнд не задан, то команда производит обмен регистров ST(0) и ST(1).

Сбрасывает в 0 признак C1 при пустом стеке.

Действие команд загрузки FLD, FILD, FBLD можно сравнить с командой push основного процессора. Аналогично ей (push уменьшает значение в регистре sp), команды загрузки сопроцессора перед сохранением значения в регистровом стеке сопроцессора вычитают из содержимого поля top регистра состояния swr единицу. Это означает, что вершиной стека становится регистр с физическим номером на единицу меньше. При этом возможно переполнение стека. Так как стек сопроцессора состоит из ограниченного числа регистров, то в него может быть записано максимум восемь значений. Из-за кольцевой организации стека, девятое записываемое значение затрет первое. Программа должна иметь возможность обработать такую ситуацию. По этой причине почти все команды, помещающие свой операнд в стек сопроцессора, после уменьшения значения поля top, проверяет регистр — кандидат на новую вершину стека — на предмет его занятости. Для анализа этой и подобных ситуаций используется регистр twr, содержащий слово тегов. Наличие регистра тегов в архитектуре сопроцессора позволяет избежать разработки программистом сложной процедуры распознавания содержимого регистров сопроцессора и дает самому сопроцессору возможность фиксировать определенные ситуации, например, попытку чтения из пустого регистра или запись в непустой регистр. Возникновение таких ситуаций фиксируется в регистре состояния swr, предназначенном для сохранения общей информации о сопроцессоре.

Команды загрузки констант

Основным назначением сопроцессора является поддержка вычислений с плавающей точкой. В математических вычислениях достаточно часто встречаются предопределенные константы. Сопроцессор хранит значения некоторых из них. Другая причина использования этих констант заключается в том, что для

определения их в памяти (в расширенном формате) требуется 10 байт, а это для хранения, например, единицы, расточительно (сама команда загрузки константы, хранящейся в сопроцессоре, занимает два байта). В формате, отличном от расширенного, эти константы хранить не имеет смысла, так как теряется время на их преобразование в тот же расширенный формат. Для каждой предопределенной константы существует своя специальная команда, которая производит загрузку ее в вершину регистрового стека сопроцессора.

Команды загрузки констант помещают одну из 7 часто используемых констант в вершину стека. Значение константы преобразуется к формату с плавающей точкой двойной повышенной точности.

Устанавливают в 1 признак `C1` при переполнении стека.

Команда `FLDZ` – загрузка нуля в вершину стека сопроцессора;
Команда `FLD1` – загрузка единицы в вершину стека сопроцессора;
Команда `FLDPI` – загрузка числа π в вершину стека сопроцессора;
Команда `FLDL2T` – загрузка $\log_2 10$ в вершину стека сопроцессора;
Команда `FLDL2E` – загрузка $\log_2 e$ в вершину стека сопроцессора;
Команда `FLDLG2` – загрузка $\lg 2$ в вершину стека сопроцессора;
Команда `FLDLN2` – загрузка $\ln 2$ в вершину стека сопроцессора;

Команды сравнения данных

Команды данной группы выполняют сравнение значений числа в вершине стека и операнда, указанного в команде.

Команды сравнения данных в вещественном формате:

Команда `F(U)COM/F(U)COMP/F(U)COMPP` – сравнение значения в вершине стека с операндом.

Синтаксис:
```
FCOM/FCOMP [источник]
FCOMPP
```

Сравнивает содержимое регистра `ST(0)` со значением операнда `источник`. По умолчанию (если операнд не задан) производит сравнение регистров `ST(0)` и `ST(1)`.В качестве операнда может быть задана ячейка памяти или регистр. Команда устанавливает биты `C0`, `C2`, `C3` регистра `swr` в соответствии с таблицей. Сбрасывает в 0 признак `C1` при опустошении стека.

Условие	C3	C2	C0
ST(0) > src	0	0	0
ST(0) < src	0	0	1
ST(0) = src	1	0	0
Недопустимая операция (#IA)	1	1	1

Команда FCOMP дополнительно выталкивает значение из ST(0).

Команда FCOMPP — сравнивает значения ST(0) и ST(1) и, после сравнения, выталкивает оба эти значения из стека.

Команда FCOMI/FUCOMI/FCOMIP/FUCOMIP – сравнение значения в вершине стека с операндом.

Синтаксис: FCOMI/FUCOMI/FCOMIP/FUCOMIP [ST(i)]

Сравнивает содержимое регистра ST(0) со значением операнда ST(i). Команда устанавливает биты ZF, PF, CF регистра EFLAGS в соответствии с таблицей. Сбрасывает в 0 признак C1 при опустошении стека.

Условие	ZF	PF	CF	Переход
ST(0) > ST(i)	0	0	0	ja
ST(0) < ST(i)	0	0	1	jb
ST(0) = ST(i)	1	0	0	je
Недопустимая операция (#IA)	1	1	1	
ST(0) >= ST(i)	(1)	0	0	jae
ST(0) <= ST(i)	(1)	0	(1)	jbe

Команда FCOMIP/FUCOMIP последним действием осуществляет выталкивание значения из ST(0).

Команда FXAM – проверка значения в вершине стека.

Синтаксис: FXAM

Проверяет содержимое регистра ST(0) и устанавливает биты C0, C2, C3 регистра swr в соответствии с таблицей. Бит C1 устанавливается равным знаковому биту ST(0).

Класс	C3	C2	C1
Неподдерживаемый формат	0	0	0
Нечисло (NaN)	0	0	1
Конечное число	0	1	0
Бесконечность	0	1	1
Ноль	1	0	0
Пустой регистр	1	0	1
Неформированное число	1	1	0

Команды сравнения данных в целочисленном формате:

Команда FICOM/FICOMP – сравнение значения в вершине стека с целочисленным операндом.

Синтаксис: `FICOM/FICOMP источник`

Сравнивает содержимое регистра `ST(0)` с целочисленным значением операнда `источник`. Длина целого операнда – 16 или 32 бита. Перед выполнением сравнения целочисленный операнд преобразуется к вещественному типу двойной расширенной точности. Команда устанавливает биты `C0`, `C2`, `C3` регистра `swr` в соответствии с таблицей 15. Устанавливает в 1 признак `C1` при переполнении стека.

Команда `FICOMP` последним действием выталкивает значения из `ST(0)`.

Команда `FTST` — сравнение значения в вершине стека с нулем.

Синтаксис: `FTST`

Команда не имеет операндов и сравнивает значения в `ST(0)` со значением `0.0` и устанавливает биты `C0`, `C2`, `C3` регистра `swr` в соответствии с таблицей 15. Сбрасывает в 0 признак `C1` при опустошении стека.

Арифметические команды

Команды сопроцессора, входящие в данную группу, реализуют четыре основные арифметические операции — сложение, вычитание, умножение и деление. Имеется также несколько дополнительных команд, предназначенных для повышения эффективности использования основных арифметических команд. С точки зрения типов операндов, арифметические команды сопроцессора можно разделить на команды, работающие с вещественными и целыми числами.

Целочисленные арифметические команды

Целочисленные арифметические команды предназначены для работы на тех участках вычислительных алгоритмов, где в качестве исходных данных используются целые числа в памяти в формате слово и короткое слово, имеющие размерность 16 и 32 бит. Перед произведением операции целочисленное значение преобразуется в вещественному формату двойной повышенной точности.

Синтаксис: `FIxxx m`

Команда FIADD – сложение ST(0)=ST(0)+m.

Команда FISUB – вычитание ST(0)=ST(0)-m.

Команда FISUBR – реверсивное вычитание ST(0)=m-ST(0).

Команда FIMUL – умножение ST(0)=ST(0)*m.

Команда FIDIV – деление ST(0)=ST(0)/m.

Команда FIDIVR – реверсивное деление ST(0)=m/ST(0).

Команды целочисленной арифметики сбрасывает в 0 признак C1 при пустом стеке, устанавливается в 1 при округлении результата.

Вещественные арифметические команды

Схема расположения операндов вещественных команд традиционна для команд сопроцессора. Один из операндов располагается в вершине стека сопроцессора — регистре ST(0), куда после выполнения команды записывается и результат. Другой операнд может быть расположен либо в памяти, либо в другом регистре стека сопроцессора. Допустимыми типами операндов в памяти являются вещественные форматы простой и двойной точности.

В отличие от целочисленных арифметических команд, вещественные арифметические команды допускают большее разнообразие в сочетании местоположения операндов и самих команд для выполнения конкретного арифметического действия.

Синтаксис:

```
Fxxx m                  ; ST(0)=ST(0)+m
Fxxx ST(0),ST(i)        ; ST(0)=ST(0)+ST(i)
Fxxx(P) ST(i),ST(0)     ; ST(i)=ST(i)+ST(0)
FxxxP                   ; ST(0)=ST(0)+ST(1)
```

Команда FADD/FADDP – сложение

Команда FSUB/FSUBP – вычитание

Команда FSUBR/FSUBRP – реверсивное вычитание

Команда FMUL/FMULP – умножение

Команда FDIV/FDIVR – деление

Команда FDIVR/FDIVRP – реверсивное деление

Команды вещественной арифметики сбрасывает в 0 признак C1 при пустом стеке, устанавливается в 1 при округлении результата.

Дополнительные арифметические команды

Команды этой группы не имеют операндов, производят действие с операндом в вершине стека сопроцессора. Результат выполнения операции сохраняется в регистре `ST(0)`. Сбрасывают в 0 признак `C1` при пустом стеке, устанавливают в 1 при округлении результата.

Команда `FSQRT` вычисление квадратного корня: $ST(0) = \sqrt{ST(0)}$.

Команда `FABS` – вычисление модуля: `ST(0)=|ST(0)|`.

Команда `FCHS` – изменение знака: `ST(0)=-ST(0)`.

Команда `FXTRACT` – выделение порядка и мантиссы. Операнд-источник по умолчанию, хранящийся в регистре `ST(0)`, разделяется на порядок и мантиссу, порядок сохраняется в `ST(0)`, а затем мантисса помещается в стек, меняя при этом указатель вершины стека (поле `top`). Для операнда, хранящего мантиссу, знак и мантисса остаются неизменными по сравнению с операндом источника. Вместо порядка записывается `3FFFh`. После выполнения команды регистр `ST(1)` хранит значение порядка исходного операнда.

Команда `FSCALE` – команда масштабирования: изменяет порядок значения, находящегося в вершине стека сопроцессора `ST(0)` на величину `ST(1)`. Команда не имеет операндов. Величина в `ST(1)` рассматривается как число со знаком. Его прибавление к полю порядка вещественного числа в `ST(0)` означает его умножение на величину $2^{ST(1)}$. С помощью данной команды удобно масштабировать на степень двойки некоторую последовательность чисел в памяти. Для этого достаточно последовательно загружать числа последовательности в вершину стека, после чего применять команду `FSCALE` и сохранять значения обратно в памяти. Команда `FSCALE` является обратной по алгоритму работы команде `FXTRACT`.

Сопроцессор имеет программно-аппаратные средства для выполнения операции округления тех результатов работы команд, которые не могут быть точно представлены. Но операция округления может быть проведена и принудительно к значению в регистре `ST(0)`, для этого предназначена последняя команда в группе дополнительных команд — команда округления.

Команда FRNDINT – округляет значение, находящееся в вершине стека сопроцессора ST(0). Команда не имеет операндов.

Возможны четыре режима округления величины в ST(0), которые определяются значениями в двухбитовом поле RC управляющего регистра сопроцессора. Для изменения режима округления используются команды FSTCWR и FLDCWR, которые, соответственно, записывают в память содержимое управляющего регистра сопроцессора и восстанавливают его обратно. Таким образом, пока содержимое этого регистра находится в памяти, вы имеете возможность установить необходимое значение поля RC.

Команда FPREM – получение частичного остатка от деления. Исходные значения делимого и делителя размещаются в стеке — делимое в ST(0), делитель в ST(1). Команда производит вычисления по формуле

$$ST(0)=ST(0)-Q*ST(1),$$

где Q – целочисленное частное от деления.

Делитель рассматривается как некоторый модуль. Поэтому в результате работы команды получается остаток от деления по модулю. Физически работа команды заключается в реализации хорошо известного всем действия: деление в столбик. При этом каждое промежуточное деление осуществляется отдельной командой FPREM. Цикл, центральное место в котором занимает команда FPREM, завершается, когда очередная полученная разность в ST(0) становится меньше значения модуля в ST(1). Судить об этом можно по состоянию флага C2 в регистре состояния swr:

- если C2=0, то работа команды fprem полностью завершена, так как разность в ST(0) меньше значения модуля в ST(1);
- если C2=1, то необходимо продолжить выполнение команды fprem, так как разность в ST(0) больше значения модуля в ST(1).

Таким образом, необходимо анализировать флаг C2 в теле цикла. Для этого C2 записывается в регистр флагов основного процессора с последующим анализом его командами условного перехода. Другой способ заключается в сравнении ST(0) и ST(1).

Команда `fprem` не соответствует последнему стандарту на вычисления с плавающей точкой IEEE-754. По этой причине в систему команд сопроцессора i387 была введена команда `fprem1`

Команда `FPREM1` – отличается от команды `FPREM` тем, что накладывается дополнительное требование на значение остатка в `ST(0)`. Это значение не должно превышать половины модуля в `ST(1)`.

После полного завершения работы команды `FPREM/FPREM1` (когда `C2=0`), биты `C0`, `C3`, `C1` содержат значения трех младших бит частного.

Команды трансцендентных функций

Сопроцессор имеет ряд команд, предназначенных для вычисления значений тригонометрических функций, таких как синус, косинус, тангенс, арктангенс, а также значений логарифмических и показательных функций.

Значения аргументов в командах, вычисляющих результат тригонометрических функций, должны задаваться в радианах. Для нахождения радианной меры угла по его градусной мере необходимо число градусов умножить на $\frac{\pi}{180}$=0,017453, число минут на $\frac{\pi}{180}\cdot 60$=0,000291, а число секунд на $\frac{\pi}{180}\cdot 60^2$=0,000005 и найденные произведения сложить.

Данная группа команд не имеет операндов. Результат сохраняется в регистре `ST(0)`. Сбрасывает в 0 признак `C1` при пустом стеке, устанавливают в 1 при округлении. Признак `C2` устанавливается в 1 при выходе значения угла за границы диапазона $-2^{63} < src < 2^{63}$.

Команда `FCOS` – вычисляет косинус угла: `ST(0)=cos(ST(0))`.

Команда `FSIN` – вычисляет синус угла: `ST(0)=sin(ST(0))`.

Команда `FSINCOS` – вычисляет синус и косинус угла: `ST(1)=sin(ST(0))`, `ST(0)=cos(ST(0))`.

Команда `FPTAN` – вычисляет частичный тангенс угла: `ST(1)=tg(ST(0))`, `ST(0)–1`. Это сделано для совместимости с сопроцессорами 8087 и 287. Выполнение данной команды в них имело следующую особенность: результат

команды возвращался в виде двух значений — в регистрах ST(0) и ST(1). Ни одно из этих значений не является истинным значением тангенса. Истинное его значение получается лишь после операции деления ST(0)/ST(1). Таким образом, для получения тангенса требовалась еще команда деления. Синус и косинус в ранних версиях сопроцессоров вычислялись через тангенс половинного угла.

$$\sin\alpha = \frac{2\cdot \operatorname{tg}\frac{\alpha}{2}}{1+\operatorname{tg}^2\frac{\alpha}{2}} \qquad \cos\alpha = \frac{1-\operatorname{tg}^2\frac{\alpha}{2}}{1+\operatorname{tg}^2\frac{\alpha}{2}}$$

В микропроцессоре i387 появились самостоятельные команды для вычисления синуса и косинуса, вследствие чего отпала необходимость составлять соответствующие подпрограммы. Что же до команды fptan, то она по-прежнему выдает два значения в st(0) и st(1). Но значение в st(0) всегда равно единице, это означает, что в st(1) находится истинное значение тангенса числа, находившегося в st(0) до выполнения команды fptan.

Команда FPATAN – вычисляет частичный арктангенс угла: $ST(0) = arctg\left(\frac{ST(1)}{ST(0)}\right)$. Команда не имеет операндов. Результат возвращается в регистр ST(1), после чего производится выталкивание вершины стека.

Команда FPATAN широко применяется для вычисления значений обратных тригонометрических функций таких, как: arcsin, arccos, arcctg, arccosec, arcsec. Например, для вычисления функции arcsin используется следующая формула:

$$\mathrm{arcsin}(a) = \mathrm{arctg}\left(\frac{a}{\sqrt{1-a^2}}\right)$$

Для вычисления этой формулы необходимо выполнить следующую последовательность шагов:

- Если a является мерой угла в градусах, то выполнить ее преобразование в радианную меру.
- Поместить в стек a в радианной мере.
- Вычислить значение выражения $\sqrt{1-a^2}$ и поместить его в стек.
- Выполнить команду fpatan с аргументами в st(0)= $\sqrt{1-a^2}$ и st(1)=a.

В результате этих действий в регистре ST(0) будет сформировано значение, которое и будет являться значением arcsin(a).

Аналогично вычисляются значения других обратных тригонометрических функций.

$$\arccos(a) = 2 \cdot \operatorname{arctg}\left(\frac{\sqrt{1-a}}{\sqrt{1+a}}\right) \qquad \operatorname{arcctg}(a) = \operatorname{arctg}\left(\frac{1}{a}\right)$$

Команда F2XM1 – вычисляет значение функции: $ST(0) = 2^{ST(0)} - 1$. Исходное значение x размещается в вершине стека сопроцессора ST(0) и должно лежать в диапазоне $-1 \le x \le +1$. Результат замещает значение в регистре ST(0). Эта команда может быть использована для вычисления показательных функций.

Единица вычитается для того, чтобы получить точный результат, когда x близок к нулю. Поскольку нормированное число всегда содержит в качестве первой значащей цифры единицу, если в результате вычисления функции 2^x получается число 1,000000000456..., то команда F2XM1, вычитая 1 из этого числа, и затем, нормируя результат, формирует больше значащих цифр, то есть делает его более точным. Неявное вычитание единицы командой F2XM1 компенсируется командой FADD с единичным операндом.

Команда FYL2X –вычисляет значение функции $ST(0) = ST(1) \cdot \log_2 ST(0)$. Значение x должно лежать в диапазоне $0 < x < +\infty$. Перед тем, как осуществить запись результата в вершину стека, команда FYL2X выталкивает значения x и y из стека, и только после этого производит запись результата в стек.

Команда FYL2XP1–вычисляет $ST(0) = ST(1) \cdot \log_2 (ST(0) + 1)$.

Значение x должно лежать в диапазоне $-\left(1 - \frac{\sqrt{2}}{2}\right) < x < \left(1 - \frac{\sqrt{2}}{2}\right)$. Перед тем, как осуществить запись результата в вершину стека, команда FYL2XP1 выталкивает значения x и y из стека, и только после этого производит запись результата в стек.

Поскольку специальной команды в сопроцессоре для операции возведения в степень нет, возведение в произвольную степень числа с любым основанием производится по формуле:

$$x^y = 2^{y \cdot \log_2 x}$$

Вычисление значения выражения $y \cdot \log_2 x$ для любых y и $x>0$ производится специальной командой сопроцессора FYL2X. Вычисление 2^x производится командой F2XM1. Лишнее действие вычитания 1 можно компенсировать сложением с единицей. Но в последнем действии есть тонкий момент, который связан с тем, что величина аргумента x должна лежать в диапазоне: $-1 \le x \le +1$. А как быть в случае, если x превышает это значение (например, для вычисления 16^3). При вычислении выражения $3 \cdot \log_2 16$ командой FYL2X, получим в стеке значение 12. Попытка вычислить значение 2^{12} командой F2XM1 ни к чему не приведет — результат будет не определен. В этой ситуации используется команда сопроцессора FSCALE, которая также вычисляет значение выражения 2^x, но для целых значений x со знаком. Применив формулу $2^{a+b} = 2^a \cdot 2^b$, получаем решение проблемы. Разделяем дробный показатель степени больший 1 по модулю на две части — целую и дробную. После этого вычисляем отдельно командами FSCALE и F2XM1 степени двойки и перемножаем результаты (не забываем компенсировать вычитание единицы в команде F2XM1 последующим сложением ее результата с единицей).

Команды управления математическим сопроцессором

Данная группа команд предназначена для общего управления работой сопроцессора. Команды этой группы имеют особенность — перед началом своего выполнения они не проверяют наличие незамаскированных исключений. Однако такая проверка может понадобиться, в частности для того, чтобы при параллельной работе основного процессора и сопроцессора предотвратить разрушение информации, необходимой для корректной обработки исключений, возникших в сопроцессоре. Поэтому некоторые команды управления имеют аналоги, выполняющие те же действия плюс одну дополнительную функцию — проверку наличия исключения в сопроцессоре. Эти команды имеют одинаковые мнемокоды (и машинные коды тоже), отличающиеся только вторым символом — символом `n`:

- мнемокод, не содержащий второго символа `n`, обозначает команду, которая перед началом своего выполнения проверяет наличие незамаскированных исключений;
- мнемокод, содержащий второй символ `n`, обозначает команду, которая перед началом своего выполнения не проверяет наличия незамаскированных исключений, то есть выполняется немедленно, что позволяет сэкономить несколько машинных тактов.

Эти команды имеют одинаковый машинный код. Отличие лишь в том, что перед командами, не содержащими символа `n`, транслятор ассемблера вставляет команду `wait`. Команда `wait` является полноценной командой основного процессора и ее, при необходимости, можно указывать явно. Команда `wait` имеет аналог среди команд сопроцессора — `fwait`. Обеим этим командам соответствует код операции `9bh`.

Команда `FWAIT` — команда ожидания. Предназначена для синхронизации работы процессора и сопроцессора. Так как основной процессор и сопроцессор работают параллельно, то может создаться ситуация, когда за командой сопроцессора, изменяющей данные в памяти, следует команда основного процессора, которой эти данные требуются. Чтобы синхронизировать работу этих команд, необходимо включить между ними команду `wait/fwait`. Встретив данную команду в потоке команд, основной процессор приостановит свою работу до

тех пор, пока не поступит аппаратный сигнал о завершении очередной команды в сопроцессоре.

Команда FINIT/FNINIT — инициализация сопроцессора. Данная команда инициализирует управляющие регистры сопроцессора определенными значениями:

CWR = 037Fh

- RC=00b; округление к ближайшему целому;
- PM,UM,OM,ZM,DM,IM=1; все исключения замаскированы;
- PC=11b; максимальная точность 64 бита.

SWR = 0h – отсутствие исключений и указание на то, что физический регистр стека сопроцессора r0 является вершиной стека и соответствует логическому регистру ST(0);

TWR = FFFFh – все регистры стека сопроцессора пусты;

DPR=0; IPR=0.

Данную команду используют перед первой командой сопроцессора в программе и в других случаях, когда необходимо привести сопроцессор в начальное состояние.

Команда FSTSW/FNSTSW — сохранение содержимого регистра состояния swr в регистре ax или в ячейке памяти размером 2 байта. Эту команду целесообразно использовать для подготовки к условным переходам по описанной при рассмотрении команд сравнения схеме.

Команда FSTCW/FNSTCW — сохранение содержимого регистра управления cwr в ячейке памяти размером 2 байта. Эту команду целесообразно использовать для анализа полей маскирования исключений, управления точностью и округления. Следует заметить, что в качестве операнда назначения не используется регистр ax, в отличие от команды FSTSW/FNSTSW.

Команда FLDCW — загрузки значения ячейки памяти размером 16 бит в регистр управления cwr. Эта команда выполняет действие, противоположное командам FSTCW/FNSTCW. Команду целесообразно использовать для задания или изменения режима работы сопроцессора. Если в регистре состояния swr установлен любой бит

исключения, то попытка загрузки нового содержимого в регистр управления cwr приведет к возбуждению исключения. По этой причине необходимо перед загрузкой регистра cwr сбросить все флаги исключений в регистре swr.

Команда FCLEX/FNCLEX — позволяет сбросить флаги исключений, которые, в частности, необходимы для корректного выполнения команды FLDCW. Ее также применяют и в случаях, когда необходимо сбрасывать флаги исключений в регистре swr, например, в конце подпрограмм обработки исключений. Если этого не делать, то при исполнении первой же команды сопроцессора прерванной программы (кроме тех команд, которые имеют в названии своего мнемокода второй символ n) будет опять возбуждено исключение.

PE, UE, OE, ZE, DE, IE, ES, SF и B биты регистра SWR равны 0.

Команда FINCSTP — увеличение указателя стека на единицу (поле top) в регистре swr. Команда не имеет операндов. Действие команды fincstp подобно команде FST, но она извлекает значение операнда из стека «в никуда». Таким образом, эту команду можно использовать для выталкивания, ставшего ненужным операнда, из вершины стека. Команда работает только с полем top и не изменяет соответствующее данному регистру поле в регистре тегов twr, то есть регистр остается занятым и его содержимое из стека не извлекается.

Команда FDECSTP — уменьшение указателя стека (поле top) в регистре swr. Команда не имеет операндов. Действие команды FDECSTP подобно команде FLD, но она не помещает значения операнда в стек. Таким образом, эту команду можно использовать для проталкивания внутрь стека операндов, ранее включенных в него. Команда работает только с полем top и не изменяет соответствующее данному регистру поле в регистре тегов twr, то есть регистр остается пустым.

Команда FFREE —помечает любой регистр стека сопроцессора как пустой.

Синтаксис: FFREE ST(i)

Команда записывает в поле регистра тегов, соответствующего регистру ST(i), значение 11b, что соответствует, пустому регистру. При этом указатель стека (поле

top) в регистре swr и содержимое самого регистра не изменяются. Необходимость в этой команде может возникнуть при попытке записи в регистр ST(i), который помечен, как непустой. В этом случае будет возбуждено исключение. Для предотвращения этого применяется команда FFREE.

Команда FNOP — пустая операция. Не производит никаких действий и влияет только на регистр указателя команды IPR.

Команда FSAVE/FNSAVE — сохранения полного состояния среды сопроцессора в память по адресу, указанному операндом приемник. Размер области памяти зависит от размера операнда сегмента кода use16 или use32:

- use 16 — область памяти должна быть 94 байта: 80 байт для восьми регистров из стека сопроцессора и 14 байт для остальных регистров сопроцессора с дополнительной информацией;
- use32 — область памяти должна быть 108 байт: 80 байт для восьми регистров из стека сопроцессора и 28 байт для остальных регистров сопроцессора с дополнительной информацией.

После выполнения сохранения состояния среды сопроцессора производится инициализация сопроцессора.

use16

CWR	0
SWR	2
TWR	4
IRP(0-15)	6
CS	8
DPR(0-15)	10
IP	12
ST(0)	14
ST(1)	24
...	
ST(7)	84

15 0

use32

0	CWR	0
0	SWR	4
0	TWR	8
IRP(0-31)		12
0	IPR(31-47)	16
DPR(0-31)		20
0	DPR(31-47)	24
ST(0)		28
ST(1)		38
...		
ST(7)		98

31 15 0

Команда FRSTOR — используется для восстановления полного состояния среды сопроцессора из области памяти, адрес которой указан операндом источник.

Сопроцессор будет работать в новой среде сразу после окончания работы команды `frstor`.

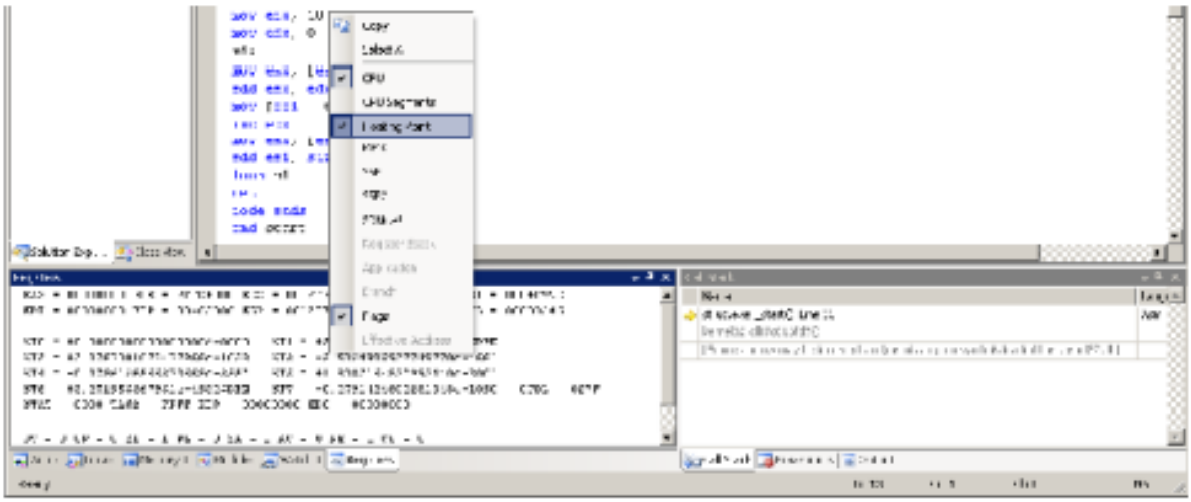
Команда `FSTENV/FNSTENV` — сохранение частичного состояния среды сопроцессора в область памяти, адрес которой указан операндом приемник. Размер области памяти зависит от размера операнда сегмента кода `use16` или `use32`. Формат области частичной среды сопроцессора совпадает с форматом области полной среды, за исключением содержимого стека сопроцессора (80 байт).

Команда `FLDENV` — восстановление частичного состояния среды сопроцессора содержимым из области памяти, адрес которой указан операндом источник. Информация в данной области памяти была ранее сохранена командой `FSTENV/FNSTENV`.

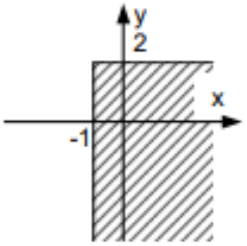
Команды сохранения среды целесообразно применять в обработчиках исключений, так как только с помощью данных команд можно получить доступ, например, к регистрам `DPR` и `IPR`. Не исключено использование этих команд при написании подпрограмм или при переключении контекстов программ в многозадачной среде.

Отладка программ, использующих функции сопроцессора

Для просмотра содержимого регистров сопроцессора в среде Microsoft Visual Studio в окне регистров выбрать всплывающее меню Floating Point.



Пример программы с использованием команд сопроцессора



Определить, принадлежит ли точка заштрихованной части плоскости.

Программа запрашивает вещественные координаты точки X и Y, после чего с помощью команды сопроцессора сравнивает координату Y с 2, а X – с (-1). В соответствии с результатом сравнения выдается сообщение о принадлежности или непринадлежности точки.

```
#include <windows.h>
#include <tchar.h>
void main()
{
float x,y;
int cons=2;
char s[40];
CharToOem(_T("Введите координаты точки X и Y: "),s);
printf(s);
scanf("%f%f",&x,&y);
_asm
     {
          fild cons
          fld y
          fcomip ST,ST(1)
          jae m
          fstp cons      ; освобождение стека
          mov cons, -1
          fild cons
          fld x
          fcomip ST,ST(1)
          jb m
     }
CharToOem(_T("Точка принадлежит плоскости."),s);
_asm jmp ext;
m:
CharToOem(_T("Точка не принадлежит плоскости."),s);
ext:
printf(s);
getch();
}
```