Языки программирования и методы трансляции
Семестр 5
«Программирование на языке ассемблера»
Часть 3

10. Программирование сопроцессора

Для обработки числовых данных в формате с плавающей точкой микропроцессор содержит специальное устройство, которое является важной частью его архитектуры.

В архитектуре семейства процессоров Intel x86 такое устройство появилось в составе компьютера на базе процессора 8086/88 и получило название математический 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 процессора 8086/88 имела название 8087. Начиная с модели 486DX, сопроцессор исполняется в одном корпусе с основным процессором и называется модулем операций с плавающей точкой (FPU, Floating Point Unit). Тем не менее, в дальнейшем мы будем использовать, хотя и устаревший, но до сих пор часто употребляемый термин «сопроцессор».

Сопроцессор рассчитан на применение в системах с интенсивной численной обработкой, в которых:

– численные данные изменяются в очень широком диапазоне или включают в себя нецелые значения;

– при реализации алгоритмов возникают очень большие или очень малые промежуточные результаты;

– требуется высокая точность вычислений с сохранением большого числа значащих разрядов;

– необходима производительность, превышающая возможности центрального процессора.

Сопроцессор помогает сократить расходы на программирование и повысить производительность систем, работающих не только с вещественными числами, но и с двоичными числами большой точности, а также с двоично-десятичными числами.

Сопроцессор расширяет математические возможности основного процессора, но не замещает ни одну из его команд. Основные арифметические команды (ADD, SUB, MUL, DIV, и другие) для целочисленных данных выполняются процессором, а математический сопроцессор выполняет дополнительные более эффективные команды арифметической обработки. С точки зрения программиста, система с математическим сопроцессором выглядит как единый процессор с расширенным набором команд.

10.1. Форматы данных

Сопроцессор специально разрабатывался для вычислений с плавающей точкой. Но он может работать и с целыми числами, хотя и менее эффективно. Перечислим форматы данных, с которыми работает сопроцессор:

– двоичные целые числа в трех форматах: 16, 32 и 64 бита;

– упакованные целые двоично-десятичные (BCD) числа – длина максимального числа составляет 18 упакованных десятичных цифр (9 байтов);

– вещественные числа в трех форматах: одинарной точности (32 бита), двойной точности (64 бита), расширенной точности (80 битов).

Кроме этих основных форматов, сопроцессор допускает *специальные численные значения*, к которым относятся:

– денормализованные вещественные числа, т.е. числа, меньшие минимального нормализованного числа для каждого вещественного формата, поддерживаемого сопроцессором;

– нуль;

– положительные и отрицательные значения бесконечности;

– нечисла;

– неопределенности и неподдерживаемые форматы.

Рассмотрим более подробно основные форматы данных, поддерживаемые сопроцессором. Важно отметить, что в самом сопроцессоре числа в этих форматах имеют одинаковое внутреннее представление – формат расширенной точности. Таким образом, даже если вы используете команды сопроцессора с целочисленными операндами, то после загрузки в сопроцессор операндов целого типа они автоматически преобразуются в формат расширенного вещественного числа.

10.1.1. Двоичные целые числа

Форматы целых чисел сопроцессора

Формат	Размер, битов	Диапазон значений
Целое слово	16	-32 768...+32 767
Короткое целое	32	$-2\cdot10^{9}...+2\cdot10^{9}$ (приблизительно)
Длинное целое	64	$-9\cdot10^{18}...+9\cdot10^{18}$ (приблизительно)

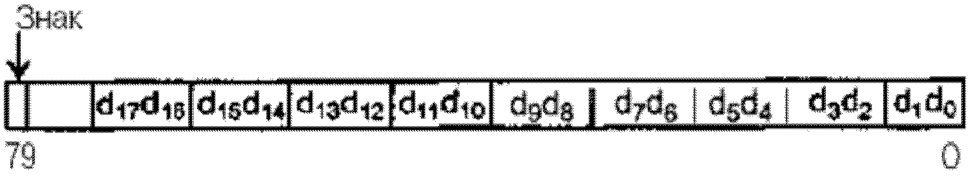
Выбирая формат данных, с которыми будет работать ваша программа, помните, что сопроцессор поддерживает операции с целыми числами, но работа с ними осуществляется неэффективно. Причина в том, что обработка сопроцессором целочисленных данных будет замедлена из-за дополнительного преобразования целых чисел в их внутреннее представление в виде эквивалентного вещественного числа расширенной точности.

В программе целые двоичные числа описываются обычным способом – с использованием директив DW, DD и DQ. Отрицательные числа хранятся в дополнительном коде.

10.1.2. Упакованные двоично-десятичные числа

Сопроцессор поддерживает один формат упакованных целых двоично-десятичных чисел, или BCD-чисел. Для описания упакованного десятичного числа используются директивы определения десяти битов DT и TBYTE. Хотя данные директивы позволяют описать 20 цифр в упакованном десятичном числе (по две в каждом байте), максимальная длина упакованного десятичного числа в сопроцессоре составляет только 9 байт, старший десятый байт игнорируется, но самый старший бит этого байта используется для хранения знака числа.

CHT DT 5365904H; представление в памяти: 04 59 36 05 00 00 00 00 00 00



Нужно отметить, что в сопроцессоре имеются всего две команды для работы с упакованными десятичными числами – это команды сохранения и загрузки.

10.1.3. Вещественные числа

Основной тип данных, с которыми работает сопроцессор – вещественный. Данные этого типа описываются тремя форматами: одинарной точности, двойной точности и расширенной точности.

Для представления вещественного числа используется следующая общепринятая формула:

$A = (\pm M) \cdot N^{\pm p}$.

Здесь:

M – мантисса числа A (мантисса должна удовлетворять условию $|M| < 1$);

N – основание системы счисления, представленное целым положительным числом;

p – порядок числа, показывающий истинное положение точки в разрядах мантиссы.

Для удобства обработки в процессоре чисел с плавающей точкой его архитектурой вносятся некоторые изменения и ограничения на элементы этой формулы.

Далее перечислены эти изменения и ограничения для сопроцессоров, применяющихся в микропроцессорах Intel.

Число представляется в следующем *нормализованном* виде

$A = (-1)^s \cdot N^q \cdot M$.

Здесь:

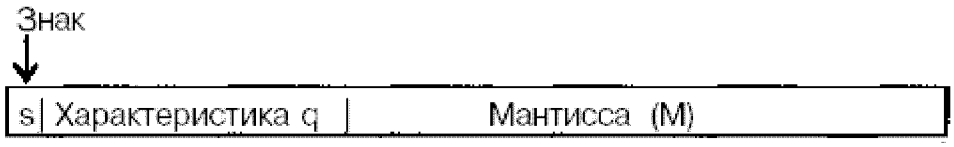
s – значение знакового разряда (0 – число больше нуля, 1 – число меньше нуля);

N=2 – основание системы счисления;

q – характеристика, связанная с порядком p посредством формулы q = p + фиксированное смещение.

M – мантисса, представленная в нормализованном виде.

Нормализованное вещественное число имеет целую единичную часть (для перечисленных ранее специальных численных значений это не всегда так), поэтому при его представлении в памяти эта единица не хранится и учитывается только на аппаратном уровне. Это дает возможность увеличить диапазон представимых чисел, так как появляется лишний разряд, пригодный для представления мантиссы числа. Но это справедливо только для вещественных чисел одинарной и двойной точности. Формат чисел расширенной точности как внутренний формат представления числа любого типа в сопроцессоре содержит целую единичную часть в явном виде.



Форматы вещественных чисел

Точность	Одинарная	Двойная	Расширенная
Длина числа (биты)	32	64	80
Размерность мантиссы M	23 + 1 = 24	52 + 1 = 53	64
Диапазон значений	$1.18 \cdot 10^{-38}$... $3.40 \cdot 10^{+38}$	$2.23 \cdot 10^{-308}$... $1.79 \cdot 10^{+308}$	$3.37 \cdot 10^{-4932}$... $1.18 \cdot 10^{+4932}$
Размерность характеристики q	8	11	15
Значение фиксированного смещения	127	1023	16383
Диапазон характеристик q	0...255	0...2047	0...32 767
Диапазон порядков p	–126...+127	–1022...+1023	–16382...+16383

Как определить вещественное число или зарезервировать место для его размещения в программе на ассемблере?

Вещественное число одинарной точности имеет длину в 32 разряда и определяется директивой DD. В записи десятичной константы при этом обязательным является наличие десятичной точки, даже если она не имеет дробной части. Для транслятора десятичная точка является указанием, что число нужно представить в виде числа с плавающей точкой в коротком формате. Это же касается форматов вещественных чисел двойной и расширенной точности, определяемых директивами DQ и DT.

Другой способ задания вещественного числа директивами DD, DQ и DT – экспоненциальная форма с использованием символа «E».

Вместо директив DD, DQ и DT можно использовать директивы REAL4, REAL8 и REAL10 соответственно.

Вещественную константу можно задать также непосредственно в форме закодированного вещественного числа в двоичном и шестнадцатеричном виде с использованием директив DD, DQ и DT; в этом случае можно использовать и директивы REALx, но только с шестнадцатеричными константами, имеющими суффикс R вместо обычного H (суффикс R вместо H можно использовать и в директивах DD, DQ, DT, но нельзя в аналогичных им по резервируемому размеру памяти DWORD, QWORD, TBYTE).

Пример. Определим в программе вещественное число 45,5 в формате одинарной точности. Это можно сделать несколькими способами:

```
DD 45.5
DD 45.5E0
DD 0.455E2
DD 0.455E+2
DD 01000010001101100000000000000000B
DD 42360000H
DD 42360000R
REAL4 42360000R
```

Учитывая, что в архитектуре Intel принят «перевернутый» порядок следования байтов в памяти в соответствии с принципом «младший байт по младшему адресу», в памяти это число будет выглядеть так:

```
00 00 36 42
```

10.1.4. Специальные численные значения

Несмотря на большой диапазон вещественных значений, представимых в сопроцессоре, понятно, что бесконечное количество их значений находится за рамками этого диапазона. Для того чтобы иметь возможность реагировать на вычислительные ситуации, в которых возникают такие значения, в сопроцессоре и предусмотрены специальные комбинации битов, называемые *специальными численными значениями*. При необходимости программист может сам кодировать специальные численные значения с использованием директив DT, REAL10, TBYTE; соответствующие команды сопроцессора загрузят их без всяких преобразований.

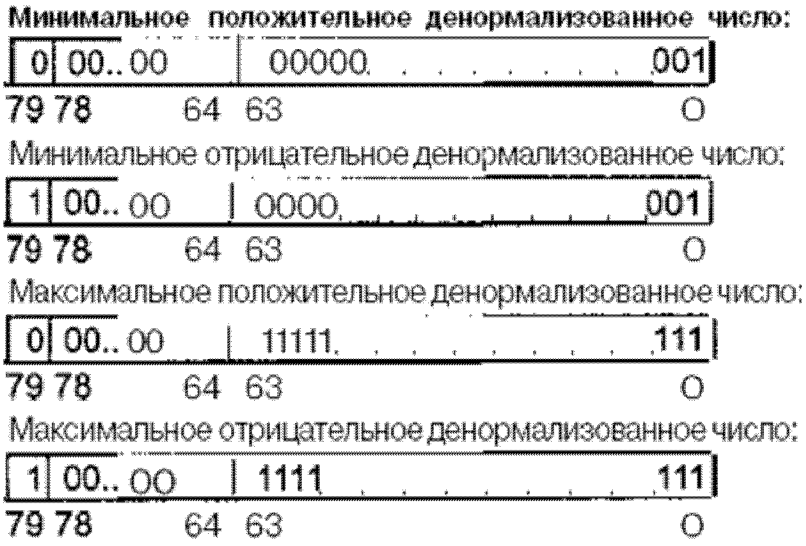
Отметим, что для представления специальных численных значений зарезервированы минимальное 00...00 и максимальное 11... 11 значения характеристики.

10.1.4.1. Денормализованные вещественные числа

В операциях могут возникать настолько малые результаты, что их характеристика должна быть отрицательной. Такая ситуация называется исчез-

новением порядка или антипереполнением. В этом случае сопроцессор сдвигает мантиссу вправо с одновременным инкрементом характеристики до тех пор, пока последняя не будет равна нулю. Такие числа с нулевой характеристикой и ненулевой мантиссой называются денормализованными. Разумеется, каждый старший нуль в мантиссе ведет к потере значимости, поэтому расширение диапазона в сторону малых чисел сопровождается потерей точности.

Таким образом, между нулем и минимально представимым нормализованным числом есть еще бесконечное количество очень маленьких денормализованных чисел. Однако, диапазон представимых в сопроцессоре денормализованных чисел не безграничен, так как количество разрядов мантиссы ограниченно.



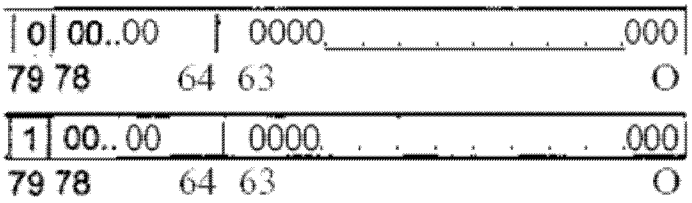
В таблице отражен приближенный диапазон денормализованных чисел.

Точность	Наименьшая величина	Наибольшая величина
одинарная	10^{-46}	10^{-38}
двойная	10^{-324}	10^{-308}
расширенная	10^{-4956}	10^{-4932}

10.1.4.2. Нуль

Нуль также относят к специальным численным значениям. Это делается из-за того, что это значение особо выделяется среди корректных вещественных значений, формируемых как результат работы некоторой команды. Более того, нуль может формироваться как реакция сопроцессора на определенную вычислительную ситуацию.

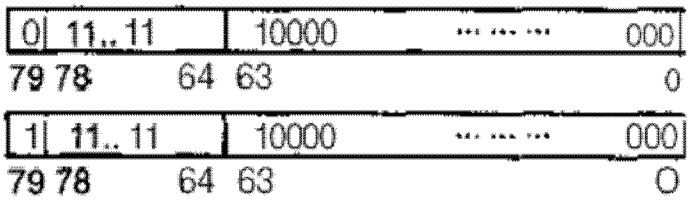
Значение истинного нуля может иметь знак, что, впрочем, не влияет на его восприятие командами сопроцессора.



Значение нуля может быть сформировано в результате возникновения ситуации антипереполнения, а также при работе команд с нулевыми операндами.

10.1.4.3. Бесконечность

Сопроцессор имеет средства в виде специальных битовых значений для представления бесконечности. Эти значения кодируются с характеристикой, состоящей из всех единиц 11... 11 и мантиссой 1.00...00.



Значение бесконечности может иметь знак, при этом значения мантиссы и характеристики фиксированы.

Среди причин, приводящих к формированию значения бесконечности, можно выделить переполнение и деление на нуль.

10.1.4.4. Нечисла

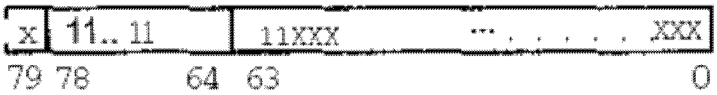
К нечислам относятся битовые последовательности следующего вида. Нечисло должно иметь характеристику, состоящую из всех единиц, и любую мантиссу, кроме 1.00...00, которая зарезервирована для значения бесконечность.

Различают два типа нечисел: SNAN (Signaling Non A Number) – сигнальные нечисла и QNAN (Quiet Non A Number) – спокойные (тихие) нечисла.

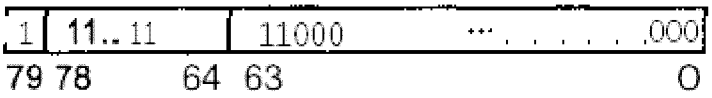
Сигнальное нечисло – битовое значение с единичными значениями битов характеристики и мантиссой, первый бит которой, следующий за первым единичным значащим битом, равен нулю.

Сопроцессор реагирует на появление этого числа возбуждением исключения недействительной операции (об исключениях подробно будет рассказано ниже). Программисты могут формировать эти числа преднамеренно, например, чтобы искусственно возбудить в нужной ситуации указанное исключение (для ускорения отладки программ). Очевидно, что именно по этой причине данные числа называются сигнальными.

Спокойное нечисло – битовое значение с единичным значением полей характеристики и мантиссой, первые два бита которой равны единице.



Сопроцессор самостоятельно не формирует сигнальных чисел, но в качестве реакции на определенные исключения он может формировать спокойные нечисла, например нечисло *вещественная неопределенность*, которое имеет вид



Неопределенность предусмотрена для вычислительных ситуаций, в которых человек говорит «не знаю». Типичным примером такой ситуации является деление 0 на 0.

Другие спокойные нечисла могут формироваться после выполнения команд, в которых хотя бы один из операндов был спокойным нечислом. Это может породить «цепную реакцию», ведущую к ошибочному результату. Поэтому в процессе вычислений рекомендуется периодически контролировать результаты исполнения команд на предмет появления спокойных нечисел.

10.1.4.5. Неподдерживаемые форматы

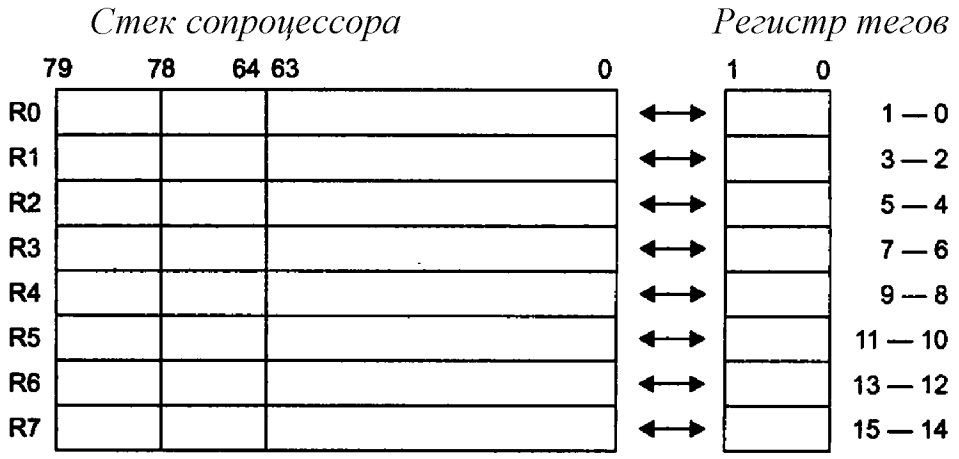
Необходимо иметь в виду, что помимо рассмотренных существует довольно много битовых наборов, которые можно представить в расширенном формате вещественного числа. Для большинства их значений формируется исключение недействительной операции.

10.2. Архитектура сопроцессора

С точки зрения программиста, сопроцессор представляет собой совокупность регистров, каждый из которых имеет свое функциональное назначение.

10.2.1. Регистровый стек

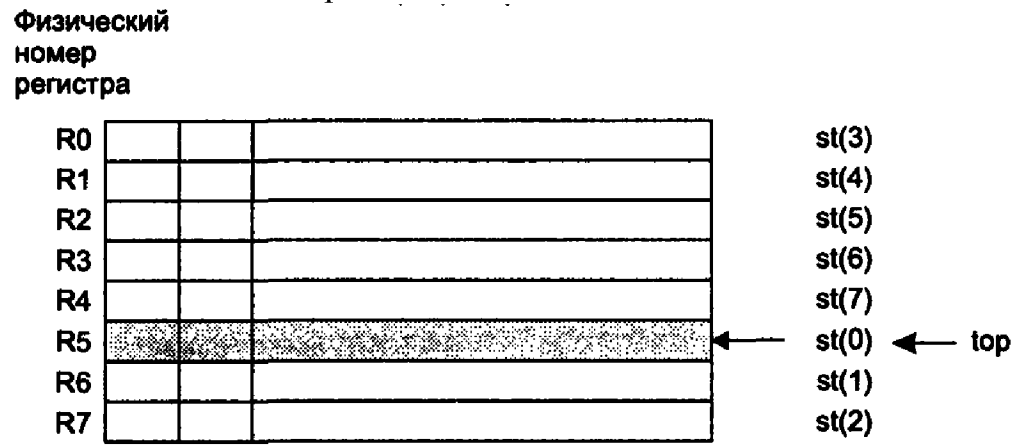
Восемь регистров R0...R7 составляют стек сопроцессора. Размерность каждого регистра составляет 80 битов – для хранения вещественных чисел расширенной точности.



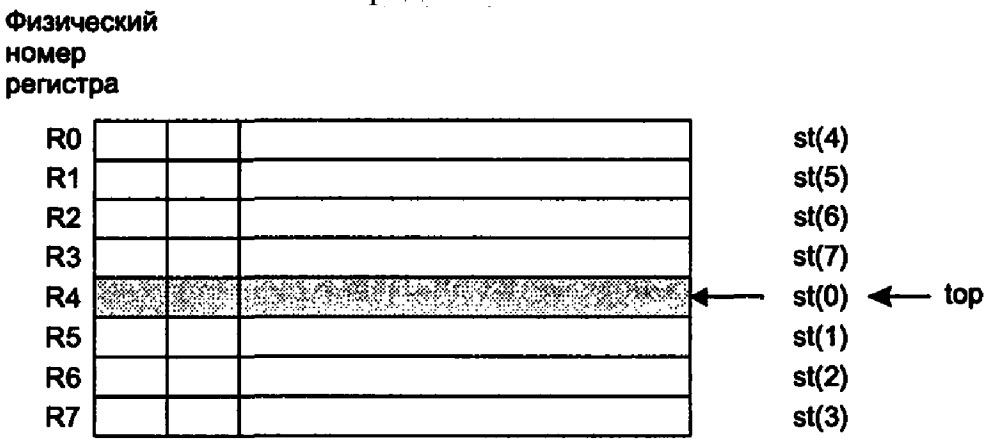
Регистровый стек сопроцессора организован по принципу кольца. Это означает, что среди всех регистров, составляющих стек, нет такого, который является вершиной стека. Напротив, все регистры стека с функциональной точки зрения абсолютно равноправны. Вершина стека является плавающей. Номер регистра стека, являющегося текущей вершиной стека, фиксируется в поле TOP регистра SWR (см. ниже).

Команды сопроцессора не оперируют физическими номерами регистров стека R0...R7. Вместо этого они используют логические номера этих регистров ST(0)...ST(7) (к регистру ST(0) можно обратиться также как к ST). С помощью логических номеров реализуется относительная адресация регистров стека сопроцессора. Если текущей вершиной стека является физический регистр R5, то после записи очередного значения в стек сопроцессора его текущей вершиной станет физический регистр R4. То есть по мере записи в стек указатель его вершины движется по направлению к младшим номерам физических регистров (уменьшается на единицу). Что касается логических номеров регистров стека ST(0).. .ST(1), то они «плавают» вместе с изменением текущей вершины стека. Таким образом, реализуется принцип кольца.

1. Стек до записи очередного значения.



2. Стек после записи очередного значения.



Подобная организация стека и выбранный метод адресации (относительно вершины стека числовых регистров) упрощает программирование сопроцессора. Во-первых, это позволяет подпрограммам передавать параметры через регистровый стек. Любая программа, выполняющая вычисления, может загрузить параметры в стек, после чего к ним можно получить доступ через логические имена регистров стека (ST(0)… ST(7)). Во-вторых, использование стека оказывается также удобным при вычислении арифметических выражений.

10.2.2. Регистр состояния

Регистр состояния сопроцессора SWR (Status Word Register) отражает информацию о текущем состоянии сопроцессора и содержит 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.

Регистр состояния SWR состоит из следующих полей.

Регистр состояния SWR

b	c3	top	c2	c1	co	es	sf	pe	ue	oe	ze	de	ie
15	14	13 11	10	9	8	7	6	5	4	3	2	1	0

Младшие шесть битов представляют собой флаги исключительных ситуаций, с помощью которых сопроцессор информирует программу о некоторых особенностях ее реального исполнения. Приведем типы исключений, фиксируемые с помощью регистра SWR:

– IE (Invalide operation Error) – недействительная операция;

– DE (Denormalized operand Error) – денормализованный 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).

При возникновении любого из этих шести типов исключений устанавливается в единицу соответствующий бит в регистре SWR.

Бит SF (Stack Fault) – ошибка работы стека сопроцессора, характер ошибки показывает бит C1. Если C1=1, произошло переполнение (команда пыталась писать в непустой регистр стека), если C1 = 0, произошло антипереполнение (команда пыталась считать число из пустого регистра стека).

Бит ES (Error Summary) сигнализирует о суммарной ошибке в работе сопроцессора. Бит устанавливается в единицу, если произошла хотя бы одна немаскированная (см. ниже) исключительная ситуация.

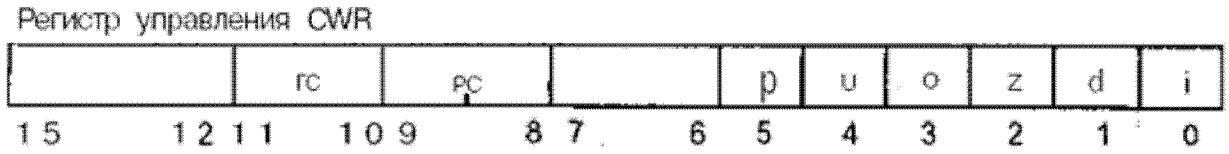
Четыре бита C0...C3 (Condition Code) представляют собой код условия. Назначение этих битов аналогично флагам в регистре EFLAGS основного процессора – они отражают результат выполнения последней команды сопроцессора.

Трехразрядное поле TOP, как мы уже знаем, содержит указатель регистра текущей вершины стека.

Бит 15 (B) всегда равен биту ES и предназначен для совместимости с сопроцессором 8087.

10.2.3. Регистр управления

Для установки режимов обработки данных сопроцессора служит регистр управления (CWR); с помощью полей в этом регистре можно регулировать точность выполнения численных вычислений, управлять округлением, маскировать исключения. Регистр управления включает в себя шесть масок исключений, поля управления точностью (precision control, PC) и поля управления округлением (rounding control, RC).



Шесть младших битов являются масками исключений, они предназначены для маскирования исключительных ситуаций, возникновение которых фиксируется в регистре состояния. Если какие-то биты исключений в регистре CWR установлены в единицу, это означает, что соответствующие исключения будут обрабатываться сопроцессором.

Если для какого-либо исключения в соответствующем бите масок исключений регистра CWR содержится нулевое значение, то обработка исключения производится специальным обработчиком, который должна содержать операционная система (или программист должен сам его написать).

Поле управления точностью PC предназначено для выбора длины мантиссы.

Возможные значения в этом поле означают:

– PC = 00 – длина мантиссы 24 бита (как у чисел одинарной точности);

– PC = 10 – длина мантиссы 53 бита (как у чисел двойной точности);

– PC = 11 – длина мантиссы 64 бита (максимальная, соответствует внутреннему формату сопроцессора).

Стандартным (см. описание команды FINIT ниже) является значение поля PC = 11. Однако, в Visual C++ по умолчанию установлено значение PC = 10.

Управление точностью влияет только на команды сложения, вычитания, умножения, деления и извлечения квадратного корня.

Поле RC позволяет управлять процессом округления чисел в ходе работы сопроцессора. Необходимость округления может возникнуть в ситуации, когда после выполнения очередной команды сопроцессора получается непредставимый результат, например периодическая дробь 3,333... Установив одно из значений в поле RC, можно выполнить округление в необходимую сторону. Для того чтобы выяснить характер округления, введем обозначения:

m – значение (результат работы некоторой команды), которое не может быть точно представлено и поэтому должно быть округлено;

a и b – наиболее близкие значения к значению m, которые могут быть представлены в регистре ST(0) сопроцессора, причем выполняется условие a<m<b.

Далее приведены значения поля RC и описан соответствующий им характер округления:

00 – значение m округляется к ближайшему числу a или b (устанавливается по умолчанию);

01 – значение m округляется в меньшую сторону, то есть m = a;

10 – значение m округляется в большую сторону, то есть m = b;

11 – производится отбрасывание дробной части m (может использоваться в операциях целочисленной арифметики).

Округление в меньшую и большую сторону называется направленным округлением и применяется в интервальной арифметике. Она позволяет определить верхнюю и нижнюю границы результата многоэтапного вычисления, когда промежуточные результаты надо округлять.

10.2.4. Регистр тегов

Регистр тегов TWR (Tags Word Register) используется для контроля за состоянием каждого из регистров R0...R7 (команды сопроцессора используют этот регистр, например, для того, чтобы определить возможность записи значений в указанные регистры).

Регистр тегов TWR представляет собой совокупность двухразрядных полей (см. рис. в п. 10.2.1). Каждое двухразрядное поле соответствует опре-

деленному 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 двухразрядных полей регистра тегов не означает, что все биты соответствующего регистра стека обязательно нулевые).

Мы уже говорили, что при написании программы разработчик манипулирует не абсолютными, а относительными номерами регистров стека. По этой причине 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.

10.3. Некоторые особенности программирования сопроцессора

10.3.1. Мнемонические обозначения команд

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 двоично-десятичное число;

отсутствие буквы – вещественное число.

Последняя буква P в мнемоническом обозначении команды означает, что последним действием команды обязательно является извлечение операнда из стека.

Последняя или предпоследняя буква R (reversed) в мнемоническом обозначении команд вычитания и деления означает реверсивное следование операндов, так как для них важен порядок следования операндов.

10.3.2. Особенности вычисления выражений

Классическая методика написания программ для сопроцессора имеет свои особенности. Главная причина здесь – в стековой организации сопроцессора. Для того чтобы написать программу для вычисления некоторого

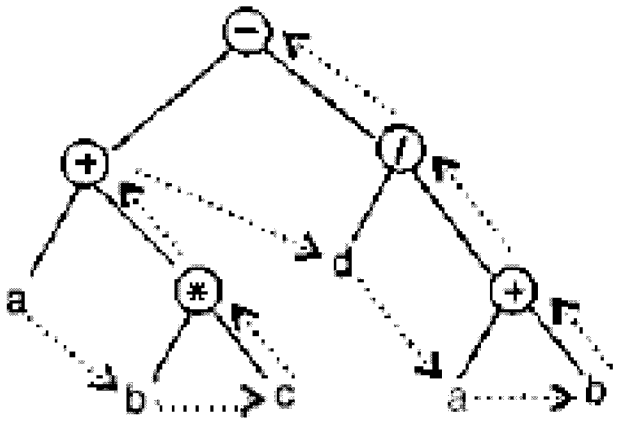
выражения, его необходимо предварительно преобразовать в удобный для программирования сопроцессора вид, а именно в обратную польскую запись.

Эта форма записи почти всегда неявно присутствует во всех алгоритмах анализа математических выражений: они сначала преобразовывают обычные выражения в обратные и только потом начинают их анализ.

Рассмотрим суть преобразования на примере вычисления выражения

a + b * c – d / (a + b)

Графически это выражение представляется в виде дерева.



Листья ветвей дерева соответствуют операндам, а узлы – операциям. Пусть началом обхода будет лист самой левой ветви дерева. Тогда для получения обратной польской записи выражения необходимо двигаться по дереву слева направо, при этом узлы должны просматриваться только после обхода всех исходящих из него ветвей. В результате выражение будет выглядеть так:

a b c * + d a b + / –

Другое название такой формы записи – постфиксная запись. Это означает, что знак операции записывается после операндов, участвующих в операции. Постфиксная запись позволяет вычислять выражения за один проход с учетом приоритета арифметических операций.

Алгоритм вычисления выражений в постфиксной записи имеет следующий вид.

1. Выбрать очередной символ постфиксной записи выражения.

2. Если очередной выбранный символ – операнд, то поместить его в стек, после чего вернуться к шагу 1.

3. Если очередной выбранный символ – знак операции, то выполнить ее с одним или двумя операндами на вершине стека (с извлечением операндов из стека). Результат операции необходимо поместить обратно на вершину стека.

4. Если в постфиксной записи выражения еще остались символы, то вернуться к шагу 1, иначе – на вершине стека получить результат вычисления выражения.

При разработке программ необходимо учитывать следующие факторы:

– ограниченность глубины стека сопроцессора;

– несовпадение форматов операндов;

– отсутствие поддержки на уровне команд сопроцессора некоторых операций, таких как возведение в степень, вычисление некоторых обратных тригонометрических функций.

Для решения этих проблем приходится разрабатывать дополнительный программный код, отклоняясь от классической (последовательной) обработки арифметических выражений, представленных в обратной польской записи.

Отметим также, что при программировании выражений необходимо следить за наполнением стека. В связи с этим рекомендуется в комментариях к командам сопроцессора указывать, какие величины или выражения содержат все используемые регистры стека.

10.3.3. Рекомендации по использованию форматов чисел

В подавляющем большинстве применений для исходных данных и результатов рекомендуется использовать формат двойной точности. Он обеспечивает достаточные для получения правильных результатов диапазон и точность, требуя от программиста минимальных усилий. Формат одинарной точности целесообразен в системах с ограничениями на память, но, конечно, он имеет меньшие диапазон и точность. Его удобно применять для отладки программ, так как ошибки округления проявляются в этом формате наиболее быстро. Формат расширенной точности не предназначен для представления входных и выходных данных, его следует применять для хранения промежуточных результатов, в циклических фрагментах и для представления констант. Его огромный диапазон и высокая точность (значительно большие, чем у базовых форматов) гарантируют защиту окончательных результатов от ошибок округления, а также уменьшают вероятность возникновения переполнений и антипереполнений. Как правило, такие ситуации свидетельствуют об ошибках в данных или программе. Формат расширенной точности обеспечивает получение результатов с 19-ю десятичными значащими цифрами. Даже когда длинная последовательность вычислений приводит к значительным накопленным ошибкам округления, потеря трех или четырех цифр точности сохраняет достаточно точный результат при представлении его в формате двойной точности.

10.3.4. Инициализация сопроцессора

Первый фрагмент программы с командами сопроцессора должен начинаться с команды инициализации сопроцессора FINIT, приводящей сопроцессор в некоторое начальное состояние. Она инициализирует управляющие регистры сопроцессора определенными значениями.

Регистр управления CWR инициализируется числом 037h, что означает установку следующих режимов:

поле округления RC = 00 – округление к ближайшему целому;

биты 0...5 установлены в единицу, что означает маскирование всех исключений;

поле управления точностью PC = 11 – максимальная точность (64 бита).

Регистр состояний SWR инициализируется нулевым значением, что означает отсутствие исключений и указание на то, что физический регистр стека сопроцессора R0 является вершиной стека и соответствует логическому регистру ST(0).

Регистр тегов TWR инициализируется единичным значением – это означает, что все регистры стека сопроцессора пусты.

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

10.4. 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 программирования сопроцессора, как команда MOV – для программирования основного процессора. С помощью команд передачи данных 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 данных к единому представлению в виде вещественного числа расширенной точности. Это же касается и обратной операции – сохранения в памяти данных из сопроцессора.

С другой стороны, действие команд загрузки можно сравнить с командой PUSH основного процессора. Аналогично ей (PUSH уменьшает значение в регистре ESP) 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 затирает 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 проверяют 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 освободить

программиста от разработки 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.

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.

10.4.1. Команды передачи данных в вещественном формате

Рассмотрим команды передачи данных в вещественном формате.

Команда загрузки вещественного числа из области памяти или из другого регистра сопроцессора на вершину стека сопроцессора имеет формат:

FLD источник

Примеры.

```
FLD MEM_DOUBLE
FLD DWORD PTR [EBX]
FLD ST(4)  ; Загрузка значения из ST(4) в ST(0).
FLD ST     ; Дублирование вершины стека ST(1) = ST(0).
```

Команды сохранения вещественного числа из вершины стека сопроцессора в память или пустой регистр стека имеют форматы:

FST приемник

FSTP приемник

Как следует из анализа мнемокода первой команды (отсутствует символ P), сохранение числа в памяти не сопровождается выталкиванием его из стека, то есть текущая вершина стека сопроцессора не меняется (поле TOP не меняется). В этой команде можно указывать переменные в памяти размером только в 32 и 64 бита.

В конце мнемонического обозначения второй команды присутствует символ P, что означает выталкивание вещественного числа из стека после его сохранения в памяти. Команда изменяет поле TOP, увеличивая его на единицу. В этой команде можно указывать переменные в памяти размером в 32, 64 и 80 битов.

Примеры.

```
FST MEM_DOUBLE
FST DWORD PTR [EBX]
FST ST(4)  ; Загрузка значения из ST(0) в ST(4).
FSTP ST ; удаление числа с вершины стека
```

10.4.2. Команды передачи данных в целочисленном формате

Рассмотрим команды передачи данных в целочисленном формате.

Команда загрузки целого числа из памяти на вершину стека сопроцессора имеет формат:

FILD источник

Команды сохранения целого числа из вершины стека сопроцессора в память имеют форматы:

FIST приемник

FISTP приемник

В первой команде сохранение числа в памяти не сопровождается выталкиванием его из стека. В этой команде можно указывать переменные в памяти размером в 16 и 32 бита.

Последним действием второй команды является выталкивание числа из стека с одновременным преобразованием его в целое значение. В этой команде можно указывать переменные в памяти размером в 16, 32 и 64 бита.

10.4.3. Команды передачи данных в двоично-десятичном формате

Команд передачи данных в двоично-десятичном формате всего две и, кроме того, это единственные команды для работы с такими данными.

Команда загрузки двоично-десятичного числа из памяти на вершину стека сопроцессора имеет формат:

FBLD источник

Следует иметь в виду, что при наличии в тетраде запрещенных комбинаций 1010B … 1111B результат загрузки двоично-десятичного операнда не определен. Другими словами, сопроцессор не контролирует правильность двоично-десятичных цифр.

Команда сохранения двоично-десятичного числа из вершины стека сопроцессора в области памяти имеет формат:

FBSTP приемник

Значение выталкивается из стека после преобразования его в формат двоично-десятичного числа.

10.4.4. Команда обмена

К группе команд передачи данных можно отнести также команду обмена вершины регистрового стека ST(0) с любым другим регистром стека сопроцессора ST(i):

FXCH ST(i)

Пример. В качестве примера определим несколько констант и выполним их пересылки между оперативной памятью и регистрами сопроцессора, а также между регистрами стека. При этом будем следить за состоянием стека сопроцессора.

```
.686
MODEL FLAT
.DATA
```

```
CH_DT DT 43567H
X DW 3
У_REAL DQ 34E7
CH_DT_ST DT 0
X_ST DW 0
Y_REAL_ST DQ 0
.CODE
MAIN PROC
FBLD CH_DT;       ST(0)=43567H
FILD X;           ST(0)=3          ST(1)=43567H
FLD У_REAL;       ST (0)=340000000 ST(1)=3      ST(2)=43567H
FXCH ST(2);       ST (0)=43567H    ST(1)=3      ST(2)=340000000
FBSTP CH_DT_ST; ST(0)=3            ST(1)=340000000
; CH_DT_ST=43567H
FISTP X_ST;       ST(0)=340000000
; X_ST=3
FSTP Y_REAL_ST
;Y_REAL_ST=340000000
RET
MAIN ENDP
END MAIN
```

10.5. Команды загрузки констант

Основным назначением сопроцессора является поддержка вычислений с плавающей точкой. В математических вычислениях довольно часто встречаются предопределенные константы, и сопроцессор хранит значения некоторых из них. Другая причина использования этих констант заключается в том, что для определения их в памяти (в формате расширенной точности) требуется 10 байт, а это для хранения, например, единицы расточительно (сама команда загрузки константы, хранящейся в сопроцессоре, занимает два байта). Заметим также, что 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 специальная команда, которая производит загрузку ее на вершину регистрового стека сопроцессора:

FLDZ – загрузка нуля;
FLD1 – загрузка единицы;
FLDPI – загрузка числа пи;
FLDL2T – загрузка двоичного логарифма десяти;
FLDL2E – загрузка двоичного логарифма экспоненты (числа *e*);
FLDLG2 – загрузка десятичного логарифма двойки;
FLDLN2 – загрузка натурального логарифма двойки.

10.6. Команды сравнения данных

Следующие команды сравнивают значение числа в вершине стека ST(0) и источника – 32- или 64-битной вещественной переменной или регистра ST(i) (если операнд не указан, то по умолчанию им является ST(1)):

FCOM [источник]

FCOMP [источник]

FCOMPP

Последним действием второй команды является выталкивание значения из ST(0). Последним действием третьей команды является выталкивание из стека значений регистров ST(0) и ST(1).

Следующие две команды сравнивают значение в вершине стека сопроцессора ST(0) с целым операндом в памяти (16- или 32-битной переменной):

FICOM источник

FICOMP источник

После сравнения вторая команда выталкивает значение из ST(0). При сравнении целые операнды преобразуются в формат расширенной точности.

Команда

FTST

не имеет операндов и сравнивает значения в ST(0) с нулем.

В результате работы команд сравнения в регистре состояния устанавливаются следующие значения битов кода условия C3, C2, C0.

Условие	C3 (ZF)	C2 (PF)	C0 (CF)
ST(0) > операнда	0	0	0
ST(0) < операнда	0	0	1
ST(0) = операнду	1	0	0
операнды неупорядочены	1	1	1

Предыдущие команды сравнения работают корректно, если операнды в них являются целыми или вещественными числами. Когда один из операндов оказывается нечислом, то фиксируется исключение недействительной операции, а коды условия C3...C0 соответствуют исключительной ситуации несравнимых или неупорядоченных операндов. Само же действие сравнения не производится.

Следующие три команды сравнивают значение числа в вершине стека ST(0) и регистра ST(i):

FUCOM [ST(i)] – команда сравнивает значения в регистрах стека сопроцессора ST(0) и ST(i);

FUCOMP [ST(i)] – команда сравнивает значения в регистрах стека сопроцессора ST(0) и ST(i); последним действием команды является выталкивание значения из вершины стека;

FUCOMPP – команда сравнивает значения в регистрах стека сопроцессора ST(0) и ST(1); последние два действия команды одинаковы – выталкивание значения из вершины стека.

Эти команды аналогичны FCOM/FCOMP/FCOMPP во всем, кроме того, что в роли источника могут выступать только регистры ST(i), и если один из операндов – QNAN (тихое нечисло), флаги C3, C2, C0 устанавливаются в единицы, но исключение недопустимой операции не вызывается. Если один из операндов – SNAN или неподдерживаемое число, эти команды ведут себя так же, как и обычное сравнение.

Для того чтобы получить возможность реагировать на результаты сравнения командами условного перехода основного процессора (вспомните, что они реагируют на флаги в EFLAGS), нужно как-то записать сформированные биты условия C3, C2, C0 в регистр EFLAGS. В системе команд сопроцессора существует команда FSTSW, которая позволяет запомнить значение регистра состояния сопроцессора SWR в регистре AX или ячейке памяти.

Далее значения нужных битов извлекаются и анализируются командами основного процессора. Обычным приемом здесь является использование команды SAHF, которая записывает содержимое старшего байта регистра AX (т.е. AH), в котором и сохраняются биты C0...C3, в младший байт регистра EFLAGS. В результате этого бит C0 записывается на место флага CF, C2 – на место PF, C3 – на место ZF (см. таблицу выше). Зная все это, вам остается, исходя из особенностей своего алгоритма, применять те команды условного перехода, которые анализируют состояние указанных флагов.

Пример. Поиск большего из двух чисел

```
.686
.MODEL FLAT
.DATA
A1 DD -162.31
A2 DD -117.03
RES DD 0.
.CODE
MAX PROC
FINIT
FLD A1 ; в стек сопроцессора загружается первое число
FLD A2 ; в стек сопроцессора загружается второе число
FCOM ; сравниваются ST(0)=A2 и ST(1) =A1
FSTSW AX ; после сравнения флаги сохраняются в регистре AX
SAHF ; флаги загружаются в регистр флагов EFLAGS
JNC STORE ; если ST(0)>ST(L), т.е. CF=0, то делается переход
FXCH ST(1) ; ST(0) и ST(1) меняются операндами и на вершине стека
           ; оказывается большее число
STORE:
FSTP RES ; число на вершине стека является максимумом и
         ; сохраняется в переменной RES
FSTP ST  ; очистка вершины стека от оставшегося аргумента
RET
MAX ENDP
END
```

В новых микропроцессорах появилась группа команд сравнения FCOMI, FCOMIP, FUCOMI, FUCOMIP, которые выполняют те же операции, что и FCOM, FCOMP, FUCOM, FUCOMP, но в отличие от последних не требуют специальных команд для обработки битов C0…C3 регистра SWR, а устанавливают флаги ZF, CF и PF непосредственно в регистре флагов EFLAGS центрального процессора. Они не изменяют содержимого регистра AX и выполняются быстрее. Формат команд (операнды обязательны):

FCOMI ST, ST(i)

Пример. В предыдущем примере заменим команду FCOM командой

FCOMI ST, ST(1)

В результате последующие две команды (FSTSW и SAHF) становятся ненужными.

К группе команд сравнения данных относят также команду FXAM, которая анализирует операнд в вершине стека сопроцессора ST(0) и формирует значение битов C0, Cl, C2, C3 в регистре состояния сопроцессора SWR. По состоянию этих битов можно судить:

о знаке мантиссы – знаковый бит операнда заносится в бит C1 регистра SWR (даже если регистр помечен как пустой);

о корректности записи операнда – идентифицируются пустой регистр, корректное вещественное число, нечисло и неизвестный формат;

о типе специального численного значения: бесконечность, нуль, денормализованный операнд.

Результаты действия команды FXAM приведены в таблице.

Тип числа	C3	C2	C0
неподдерживаемое	0	0	0
не-число	0	0	1
нормальное конечное число	0	1	0
бесконечность	0	1	1
ноль	1	0	0
регистр пуст	1	0	1
денормализованное число	1	1	0

После выполнения команды FXAM содержимое регистра состояния сопроцессора SWR для его анализа удобно переслать в регистр AX командой FSTSW.

10.7. Пересылка по условию

В систему команд новых микропроцессоров включены команды, которые являются аналогами команд CMOVx и которые можно обозначить как FCMOVx, где x – мнемоническое обозначение кода условия. Формат команд:

FCMOVx ST, ST(i)

Эти команды выполняют копирование регистра ST(i) в регистр ST(0), если выполняется указанное в команде условие.

Названия команд и анализируемые ими флаги показаны в таблице.

Мнемоническое обозначение	Состояние флагов	Описание условия
FCMOVB	CF = 1	меньше
FCMOVNB	CF = 0	не меньше
FCMOVE	ZF = 1	равно
FCMOVNE	ZF = 0	не равно
FCMOVBE	CF = 1 или ZF = 1	меньше или равно
FCMOVNBE	CF = 0 и ZF = 0	не меньше или равно
FCMOVU	PF = 1	несравнимы
FCMOVNU	PF = 0	сравнимы

Так же как и команды CMOVx, команды FCMOVx хорошо подходят для оптимизации вычислительных алгоритмов, исключая избыточные ветвления и переходы в программе.

Пример. Модифицируем еще раз процедуру поиска большего из двух чисел

```
.686
.MODEL FLAT
.DATA
A1 DD -162.31
A2 DD -117.03
RES DD 0.
.CODE
MAX PROC
FINIT
FLD A1 ; в стек сопроцессора загружается первое число
FLD A2 ; в стек сопроцессора загружается второе число
FCOMI ST, ST(1) ; сравниваются ST(0)=A2 и ST(1) =A1
FCMOVB ST, ST(1) ; если ST(0)<ST(1), то CF=1 и ST(1)
                 ; записывается на вершину стека
FSTP RES ; число на вершине стека является максимумом и
         ; сохраняется в переменной RES
FSTP ST  ; очистка вершины стека от оставшегося аргумента
RET
MAX ENDP
END
```

10.8. Арифметические команды

Команды сопроцессора, входящие в группу арифметических команд, 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. Рассмотрим их.

10.8.1. Целочисленные арифметические команды

Целочисленные арифметические команды предназначены для работы на тех участках вычислительных алгоритмов, где в качестве исходных данных используются целые числа в памяти длиной 16- и 32-битов. В целочисленных арифметических командах источником всегда является такое целое число, а приемником – регистр ST(0).

Действие следующих команд отразим формулой

Приемник = Приемник *операция* Источник

Команда сложения значения ST(0) и источника:

FIADD источник

Команда вычитания значения источника из ST(0):

FISUB источник

Команда умножения значения источника на ST(0):

FIMUL источник

Команда деления ST(0) на значение источника:

FIDIV источник

Реверсивные команды действуют согласно формуле

Приемник = Источник *операция* Приемник

Реверсивная команда вычитания значения ST(0) из источника:

FISUBR источник

Реверсивная команда деления значения источника на ST(0):

FIDIVR источник

Пример. Рассмотрим пример использования целочисленных арифметических команд. Предположим, необходимо вычислить выражение (A + B)/(A – B), где A и B – целые числа. Процедура возвращает округленное до ближайшего целого числа значение, если параметры A и B не равны, или 0 в случае равенства операндов.

```
.686
.MODEL FLAT
.DATA
OP DD 0
RES DD 0
A DW 22
B DW 12
.CODE
INTOPS PROC
FINIT
FILD A ; загрузка A в стек, ST(0)=A
FICOM B ; сравнение A и B
FSTSW    AX
```

```
SAHF
JZ          EQ_0 ; если A и B равны, то результат нулевой - RES = 0
; если не равны
FISUB    B ; ST(0) = A - B
FISTP    OP ; OP = A - B
FILD      A ; ST(0) = A
FIADD    B ; ST(0) = A + B
FIDIV    OP ; ST(0) = (A + B)/(A - B)
FISTP RES ; RES = (A + B)/(A - B)
EQ_0:
RET
INTOPS ENDP
END
```

10.8.2. Вещественные арифметические команды

Схема расположения операндов вещественных команд традиционна для команд сопроцессора. Один из операндов располагается в вершине стека сопроцессора – регистре ST(0), а второй операнд может быть расположен либо в памяти, либо в другом регистре стека сопроцессора. Допустимыми типами операндов в памяти являются все перечисленные ранее вещественные форматы за исключением расширенного.

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

Общий формат команд также традиционен:

Код_команды Приемник, Источник

Традиционные команды действуют по формуле

Приемник = Приемник *операция* Источник

Реверсивные команды для вычитания и деления действуют по формуле

Приемник = Источник *операция* Приемник

Форматы команд, их действия и примеры приведены в таблицах. В них обозначено: x – модификатор команды: ADD для сложения, SUB для вычитания, MUL для умножения, DIV для деления, Mem – источник в памяти. Для команд вычитания и деления допустима буква R определяющая реверсивное действие команды. Буква P в конце названия определяет завершающее действие команды в виде удаления операнда-источника из стека.

Традиционные команды

Формат команды	Действия	Пример
Fx ST, ST(i)	ST(0) = ST(0) *операция* ST(i)	FADD ST, ST(3)
Fx ST(i), ST	ST(i) = ST(i) *операция* ST(0)	FDIV ST(2), ST

FxP ST(i), ST	ST(i) = ST(i) *операция* ST(0) после удаления источника из стека результат оказывается в ST(i-1)	FMULP ST(4), ST
Fx Mem	ST(0) = ST(0) *операция* Mem	FADD BETA
Fx FxP	ST(1) = ST(1) операция ST(0) после удаления источника из стека результат оказывается в ST(0)	FSUB FSUBP

Реверсивные команды

Формат команды	Действия	Пример
FxR ST, ST(i)	ST(0) = ST(i) *операция* ST(0)	FSUBR ST, ST(3)
FxR ST(i), ST	ST(i) = ST(0) *операция* ST(i)	FDIVR ST(2), ST
FxRP ST(i), ST	ST(i) = ST(0) *операция* ST(i) после удаления источника из стека результат оказывается в ST(i-1)	FDIVRP ST(4), ST
FxR Mem	ST(0) = Mem *операция* ST(0)	FSUBR GAMMA
FxR FxRP	ST(1) = ST(0) *операция* ST(1) после удаления источника из стека результат оказывается в ST(0)	FSUBR FSUBRP

Пара команд без операндов Fx и FxP производит идентичные действия и совпадает по действию с командой

FxP ST(1), ST

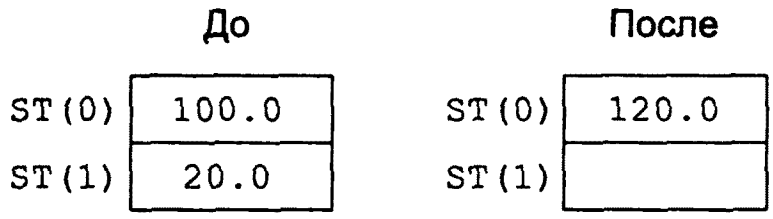
Аналогично, пара реверсивных команд без операндов FxR и FxRP совпадает по действию с командой

FxRP ST(1), ST

Эти команды называются классическими стековыми командами. Они выполняют операции над содержимым регистров, находящихся на вершине стека: ST(1) – приемник, ST(0) – источник. После выталкивания источника из стека результат оказывается на его вершине – в регистре ST(0).

Пример. Команды FADD и FADDP прибавляют содержимое ST(0) к ST(1), помещают результат в ST(1) и удаляют содержимое ST(0) из стека, так что результат оказывается в ST(0).

```
.DATA
OP1 REAL4 20.0
OP2 REAL4 100.0
.CODE
FLD OPL ;  ST(0)=20.0
FLD OP2 ;  ST(0)=100.0, ST(1)=20.0
FADD  ;    ST(0)=120.0
```



Результат выполнения команды FADD

Для вычисления выражений, представленных в постфиксной форме, строго в соответствии с описанным выше алгоритмом лучше всего подходят именно классические стековые команды.

Пример. Вычислим выражение

a + b * c – d / (a + b)

для которого ранее была найдена постфиксная форма

a b c * + d a b + / –

Рассмотрим программу, реализующую эти вычисления (вместо переменной *c* используется переменная *z*).

;	ST(0)	ST(1)	ST(2)	ST(3)
FLD A ;	A			
FLD B ;	B	A		
FLD Z ;	Z	B	A	
FMULP ;	B*Z	A		
FADDP ;	A+B*Z			
FLD D ;	D	A+B*Z		
FLD A ;	A	D	A+B*Z	
FLD B ;	B	A	D	A+B*Z
FADDP ;	B+A	D	A+B*Z	
FDIVP ;	D/(B+A)	A+B*Z		
FSUBP ;	A+B*Z-D/(B+A)			

В то же время такая схема вычислений не является оптимальной: аргументы a и b дважды загружаются в стек из памяти, что к тому же каждый раз сопровождается преобразованием форматов, если, конечно, a и b не хранятся в памяти в расширенном формате. Используя иные последовательности действий и команды, отличные от классических стековых, можно оптимизировать вычисления.

Например, возможна следующая схема вычислений.

;	ST(0)	ST(1)	ST(2)
FLD A ;	A		
FLD B ;	B	A	
FLD Z ;	Z	B	A
FMUL ST, ST(1);	B*Z	B	A
FADD ST, ST(2);	A+B*Z	B	A
FXCH ST(2);	A	B	A+B*Z
FADDP ;	A+B	A+B*Z	
FLD D ;	D	A+B	A+B*Z

```
FDIVRP          D/(B+A)             A+B*Z
FSUBP ;         A+B*Z-D/(B+A)
```

10.8.3. Дополнительные арифметические команды

Рассмотрим команды, входящие в группу дополнительных арифметических команд.

FSQRT – команда вычисления квадратного корня из значения, находящегося в вершине стека сопроцессора – регистре ST(0). Команда не имеет операндов. Результат вычисления помещается в регистр ST(0). Следует отметить, что данная команда обладает определенными достоинствами. Во-первых, результат извлечения достаточно точен, во-вторых, скорость исполнения чуть больше скорости выполнения команды деления вещественных чисел FDIV.

FABS – команда вычисления модуля значения, находящегося в вершине стека сопроцессора – регистре ST(0). Команда не имеет операндов. Результат вычисления помещается в регистр ST(0);

FCHS – команда изменения знака значения, находящегося в вершине стека сопроцессора – регистре ST(0). Команда не имеет операндов. Результат вычисления помещается обратно в регистр ST(0).

FXTRACT – команда выделения порядка и мантиссы значения, находящегося в вершине стека сопроцессора – регистре ST(0). Команда не имеет операндов.

Результат выделения помещается в два регистра стека – мантисса в ST(0), а порядок в ST(1). При этом мантисса представляется вещественным числом с тем же знаком, что и у исходного числа, и порядком равным нулю. Порядок, помещенный в ST(1), представляется как истинный порядок, то есть без константы смещения, в виде вещественного числа со знаком.

На последнюю команду следует обратить особое внимание, так как она позволяет выделить и затем по отдельности проанализировать порядок и мантиссу числа, содержащегося в вершине стека.

Восстановить исходное значение числа, преобразованного командой FXTRACT, можно следующей командой.

FSCALE – команда масштабирования. Она изменяет порядок значения, находящегося в вершине стека сопроцессора – регистре ST(0), на величину в 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 $2^{ST(1)}$, то есть $ST(0) = ST(0) \cdot 2^{ST(1)}$.

Если значение ST(1) не является целым числом, команда FSCALE использует ближайшее целое, меньшее по величине; т. е. она усекает значение масштаба к 0. Если полученное целое равно 0, значение в ST не изменяется.

Команда позволяет производить быстрое умножение на целочисленную степень числа 2.

Команда FSCALE по алгоритму работы является обратной команде FXTRACT.

Пример. Работа команд FXTRACT и FSCALE для числа $45.56=1.42375\cdot2^5$

```
.686
.MODEL FLAT
.DATA
Y1 REAL10 45.56
MANT REAL4 0.
ORD REAL4 0.
Y2 REAL10 0.
.CODE
EXTRACT PROC
FLD Y1 ; ST(0) = 45.56
FXTRACT ; ST(0) = 1.42375,  ST(1) = 5.
FST MANT ; MANT = 1.42375, ST(0) = 1.42375,  ST(1) = 5.
FXCH ; ST(0) = 5., ST(1) = 1.42375
FST ORD ; ORD = 5., ST(0) = 5., ST(1) = 1.42375
FXCH ; ST(0) = 1.42375,  ST(1) = 5.
FSCALE ; ST(0) = 45.56,  ST(1) = 5.
FSTP Y2; Y2 = 45.56,  ST(0) = 5.
FSTP ST ; очистка стека от порядка числа
RET
EXTRACT ENDP
END
```

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

FRNDINT – команда округления. Она округляет до целого значение, находящееся в вершине стека сопроцессора – регистре ST(0). Команда не имеет операндов.

Возможны четыре режима округления величины в ST(0), которые определяются значениями в двухразрядном поле RC регистра управления сопроцессора CWR (биты 10 и 11).

Как изменить режим округления? С помощью двух команд FSTCW и FLDCW, которые, соответственно, записывают в память содержимое регистра управления сопроцессора CWR и восстанавливают его обратно. Таким образом, пока содержимое этого регистра находится в памяти, можно установить необходимое значение поля RC.

Следует отметить, что 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. Для этого предназначена команда FCLEX (она также проверяет наличие произошедших и необработанных исключений и обрабатывает их до выполнения).

Пример.

```
.686
.MODEL FLAT
.DATA
MEM16 DW 0
Y_REAL DT 10.0
X_3 DD 3.0
.CODE
MAIN PROC
FLD Y_REAL
FLD X_3
FDIV ; ST(0)=3.333...33
FSTCW MEM16  ; сохранение CWR в памяти
AND MEM16, 1111101111111111B ; установка RC = 10 - округление
OR    MEM16, 0000100000000000B ; в большую сторону
FLDCW MEM16 ; загрузка из памяти в CWR
FRNDINT ; ST(0)=4
RET
MAIN ENDP
END MAIN
```

FPREM – команда получения частичного остатка от деления. Исходные значения делимого и делителя размещаются в стеке – делимое в ST(0), делитель в ST(1). Делитель рассматривается как некоторый модуль. Поэтому в результате работы команды получается остаток от деления по модулю. Но произойти это может не сразу, так как этот результат в общем случае достигается за несколько последовательных обращений к команде FPREM, если значения операндов сильно различаются.

Эта команда предназначена, в основном, для приведения аргумента периодических функций в диапазон, допустимый в командах сопроцессора.

Физически работа команды осуществляется при помощи последовательных вычитаний ST(1) из ST(0), но за один раз выполняется не более 64 таких вычитаний. Если ST(0) не стал меньше ST(1) за это время, говорят, что в ST(0) находится *частичный остаток* от деления. Если был получен точный остаток, флаг C2 сбрасывается в 0, если частичный – устанавливается в 1, так что можно повторять эту команду до обнуления C2.

Для анализа флага C2 в теле цикла он записывается в регистр флагов основного процессора с последующим анализом его командами условного перехода. Другой способ контроля выполнения команды заключается в сравнении ST(0) и ST(1).

Необходимость в таком частичном исполнении команды FPREM возникает из-за того, что если делимое слишком велико, а делитель мал, то время получения конечного частичного остатка, удовлетворяющего условию ST(0) < ST(1), может быть достаточно большим. Чтобы предотвратить «зависание» процессора, команда FPREM рассчитана на итеративное выполнение

в программном цикле. За одну итерацию при выполнении команды FPREM значение в ST(0) может уменьшиться в 2^{64} раз.

Важно отметить, то команда FPREM не соответствует стандарту IEEE-754 на вычисления с плавающей точкой. По этой причине в систему команд сопроцессора введена команда FPREM1, которая 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). В остальном работа команды FPREM1 аналогична работе FPREM.

10.9. 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. Наличие этих команд значительно облегчает жизнь программисту, вынужденному интенсивно заниматься разработкой вычислительных алгоритмов.

Операнды команд находятся в одном или двух верхних регистрах стека, и результат также возвращается в стек. Предполагается, что операнды нормализованы и находятся в допустимом для каждой из команд диапазоне. При нарушении этих требований результат операции не определен; более того, сопроцессор не оповещает об этом никаким исключением.

Результаты трансцендентных команд имеют очень высокую точность. Абсолютное значение относительной ошибки в них меньше 2^{-62}.

Тригонометрические функции допускают практически неограниченный диапазон операндов, а другие трансцендентные команды требуют нахождения аргументов в более ограниченном диапазоне.

10.9.1. Команды тригонометрических функций

Прежде всего необходимо обратить внимание на то, что значения аргументов в командах, вычисляющий результат тригонометрических функций, должны задаваться в радианах. В связи с этим приведем правило пересчета. Для нахождения радианной меры угла по его градусной мере необходимо число градусов умножить на $\pi/180$ ($\approx$0,017453), число минут – на $\pi/(180\cdot60)$ ($\approx$0,000291), а число секунд – на $\pi/(180\cdot60\cdot60)$ ($\approx$0,000005) и найденные произведения сложить.

Рассмотрим команды трансцендентных функций.

FCOS – команда вычисляет косинус угла, находящегося в вершине стека сопроцессора – регистре ST(0). Команда не имеет операндов. Результат возвращается в регистр ST(0). Аргумент не может быть больше 2^{63} и меньше -2^{63}.

FSIN – команда вычисляет синус угла, находящегося в вершине стека сопроцессора – регистре ST(0). Команда не имеет операндов. Результат возвращается в регистр ST(0). Аргумент не может быть больше 2^{63} и меньше -2^{63}.

FSINCOS – команда вычисляет синус и косинус угла, находящегося в вершине стека сопроцессора – регистре ST(0). Команда не имеет операндов. Результат возвращается в регистрах ST(0) и ST(1). При этом синус помещается в ST(1), а косинус – в ST(0). Аргумент не может быть больше 2^{63} и меньше -2^{63}.

FPTAN – команда вычисляет тангенс угла, находящегося в вершине стека сопроцессора – регистре ST(0). Команда не имеет операндов. Результат возвращается в регистре ST(1), в регистр ST(0) помещается единица. Такие действия команда производит ради совместимости с ранними моделями сопроцессора, однако, это позволяет вычислять котангенс парой команд FPTAN, FDIVR. Аргумент не может быть больше 2^{63} и меньше -2^{63}.

В командах FSIN, FCOS, FSINCOS и FPTAN, если операнд в вершине стека лежит вне допустимого диапазона, устанавливается бит C2, а в вершине стека сохраняется исходный операнд.

Для приведения операнда в допустимый диапазон можно воспользоваться командой FPREM. Для команд вычисления синуса и косинуса следует воспользоваться FPREM с делителем 2π, а для команды вычисления тангенса – с делителем π.

FPATAN – команда вычисляет арктангенс числа, получаемого при делении ST(1) на ST(0), сохраняет результат в ST(1) и выталкивает ST(0) из стека. Результат всегда имеет тот же знак, что и ST(1), и меньше π по абсолютной величине. Команда не имеет операндов.

FPATAN может выполняться над любыми операндами (кроме нечисел), давая результаты для различных нулей и бесконечностей, как показано в таблице. F в этой таблице – конечное вещественное число.

ST(1) \ ST(0)	-∞	-F	0	0	+F	+∞
-∞	$-3\pi/4$	$-\pi/2$	$-\pi/2$	$-\pi/2$	$-\pi/2$	$-\pi/4$
-F	$-\pi$	от $-\pi$ до $-\pi/2$	$-\pi/2$	$-\pi/2$	от $-\pi/2$ до -0	0
0	$-\pi$	$-\pi$	$-\pi$	0	0	0
0	$+\pi$	$+\pi$	$+\pi$	0	0	0
+F	$+\pi$	от $+\pi$ до $+\pi/2$	$+\pi/2$	$+\pi/2$	от $+\pi/2$ до +0	0
+∞	$+3\pi/4$	$+\pi/2$	$+\pi/2$	$+\pi/2$	$+\pi/2$	$+\pi/4$

Команда FPATAN широко применяется для вычисления значений других обратных тригонометрических функций, таких как arcsin, arccos, arcctg, arccosec, arcsec. В частности, используются формулы

arcsin(a) = arctg(a/sqrt(1 – a^2)),

arccos(a) = 2·arctg(sqrt(1 – a)/sqrt(1 + a)),

arcctg(a) = arctg(1/a).

Пример. Вычисление функции arcsin(a).

;	ST(0)	ST(1)	ST(2)
FLD A ;	A		
FLD ST ;	A	A	
FMUL ST, ST(1) ;	A*A	A	
FLD1 ;	1	A*A	A
FSUBRP ;	1-A*A	A	

FSQRT ;	sqrt(1-A*A)	A
FPATAN ;	arctg(A/sqrt(1-A*A))	

10.9.2. Команды вычисления логарифмов и степеней

F2XM1 – команда вычисления значения функции $y = 2^x - 1$. Исходное значение x размещается в вершине стека сопроцессора в регистре ST(0) и должно лежать в пределах $-1 \le x \le 1$. Результат замещает x в регистре ST(0). Эта команда может быть использована для вычисления различных показательных функций.

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

FYL2X – команда вычисления значения функции $z = y \cdot \log_2(x)$. Исходное значение x размещается в вершине стека сопроцессора и должно быть неотрицательным ($0 \le x \le +\infty$), исходное значение y размещается в регистре ST(1). Перед тем как осуществить запись результата z в вершину стека, команда FYL2X выталкивает значения x и y из стека.

Команды сопроцессора F2XM1 и FYL2X очень полезны, в частности, для реализации операции возведения в степень любого числа по любому основанию. Специальной команды в сопроцессоре для возведения в степень нет. Поэтому программисту нужно искать подходящую формулу приведения, которая позволит реализовать отсутствующую операцию имеющимися программно-аппаратными средствами сопроцессора. 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 любого y и неотрицательного x производится командой сопроцессора FYL2X. Вычисление 2^x производится командой F2XM1 (лишнее вычитание единицы всегда 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, если необходимо вычислить 16^3, то при вычислении выражения $3 \cdot \log_2(16)$ командой FYL2X мы получим в стеке значение 12. В этой ситуации логично воспользоваться командой 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, по модулю больший 1, на две части – целую и дробную. После этого вычисляем отдельно командами FSCALE и F2XM1 степени двойки и пере-

множаем результаты (не забываем компенсировать вычитание единицы в команде F2XM1 последующим сложением результата с единицей).

Используя тождества $\log_n(x) = \log_2(x)\cdot\log_n(2) = \log_2(x)/\log_2(n)$, можно вычислить логарифм любого числа по любому основанию. Значения констант lg(2), ln(2), $\log_2(10)$, $\log_2(e)$ загружаются соответствующими командами сопроцессора: FLDLG2, FLDLN2, FLDL2T, FLDL2E.

FYL2XP1 – команда вычисления значения функции $z=y\cdot\log_2(x+1)$. Исходное значение x размещается в вершине стека сопроцессора – регистре ST(0), а исходное значение y – в регистре ST(1). Значение x должно лежать в диапазоне $0 \le |x| \le 1-\sqrt{2}/2$. Перед тем как записать результат z в вершину стека – регистр ST(0), команда FYL2XP1 выталкивает из стека значения x и y. Данная команда дает бóльшую точность для ST(0), близких к нулю, чем FYL2X для суммы того же ST(0) и 1.

Пример. Вычисление натурального логарифма ln(x).

;	ST(0)	ST(1)
FLD1 ;	1	
FLD X ;	X	1
FYL2X;	log2(x)	
FLDL2E ;	log2(e)	log2(x)
FDIVP ;	ln(x)	

10.10. Команды управления сопроцессором

В системе команд сопроцессора имеется большая группа, предназначенная для общего управления работой сопроцессора. Все эти команды обычно требуются при разработке программ с обработчиками исключительных ситуаций, а также в многозадачной среде, хотя могут использоваться и в обычных приложениях для создания оригинальных алгоритмов обработки чисел. Часть этих команд, работающих с регистрами SWR и CWR, мы уже рассмотрели.

Рассмотрим несколько команд, управляющих состоянием стека.

Сопроцессор имеет две команды, которые работают с указателем стека в регистре SWR.

FINCSTP – команда увеличения указателя стека на единицу (поле TOP) в регистре SWR. Команда не имеет операндов. Действие команды FINCSTP подобно действию команды 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.

FDECSTP – команда уменьшения указателя стека (поле TOP) в регистре SWR. Команда не имеет операндов. Действие команды FDECSTP подобно действию команды 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 ST(i), помечает любой регистр стека сопроцессора как пустой. Это команда освобождения регистра стека ST(i). Команда записывает в поле регистра тегов, соответствующего регистру ST(i), значение 11, 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 предотвращений этого применяется команда FFREE.

10.11. Исключения сопроцессора и их обработка

Важной особенностью сопроцессора является возможность распознавания особых ситуаций – исключений, которые могут возникать в процессе вычислений. При этом сопроцессор устанавливает соответствующие биты в своем регистре состояния SWR.

В сопроцессоре могут возникать шесть типов исключений. Все они были перечислены при обсуждении форматов регистра состояния SWR и регистра управления CWR.

Вспомним основные моменты. В регистре состояния SWR 6 битов, каждый из которых играет роль флага для определенного типа исключения. При возникновении исключения устанавливается соответствующий флаг в этом регистре. Сопроцессор аппаратно позволяет запретить явную обработку любого из этих типов исключений. Для этого в регистре управления CWR есть 6 битов, играющих роль масок, установка которых и позволяет запретить возникновение соответствующих исключений. Важно отметить, что запрещение обработки исключения вовсе не означает того, что сопроцессор оставляет вычислительную ситуацию неизменной. При возникновении исключения, имеющего единичное состояние соответствующего бита в регистре CWR, сопроцессор выполняет так называемую маскированную реакцию, которая включает в себя некоторую последовательность предопределенных разработчиками процессора действий. Такие маскированные реакции безопасны и приемлемы для большинства численных приложений. Естественно, что в них невозможно было предусмотреть все потребности программистов, которым, помимо всего прочего, могут понадобиться и нестандартные варианты реакции на исключения. Разработка обработчиков исключений довольно сложна, поэтому мы ее рассматривать не будем. Мы рассмотрим лишь причины возникновения исключений и маскированные реакции на них со стороны сопроцессора. Эта информация может быть полезна для того, чтобы разобраться с логическими ошибками программы и правильно запрограммировать реакцию на их возникновение.

10.11.1. Недействительная операция

Причиной исключения недействительной операции являются ошибки в логике программы, наиболее типичные из которых следующие:

– загрузка операнда в непустой регистр стека сопроцессора;

– попытка извлечения операнда из пустого регистра стека сопроцессора;

– использование операнда с недопустимым для данной операции значением.

При возникновении этого исключения устанавливается флаг IE (Invalid Operation) в регистре SWR, маскируется оно битом IM регистра управления CWR. Исключение недействительная операция возникает при работе со стеком и при арифметических вычислениях. В каких именно операциях возникло это исключение, судят по содержимому бита SF (Stack Fault – ошибка стека) регистра состояния SWR. Если он установлен в единицу, это говорит о том, что исключение было вызвано ошибкой в работе стека сопроцессора; напротив, нулевое состояние бита SF говорит о том, что в команде встретился неверный операнд.

Маскированная реакция на исключение недействительной операции зависит от причины ошибки. Если она возникла в результате некорректной работы стека, то сопроцессор перезаписывает содержимое того регистра стека, обращение к которому вызвало исключение. При перезаписи в него заносится специальное численное значение – вещественная неопределенность. Если ошибка возникла в результате недопустимого операнда, то в большинстве ситуаций в регистре стека сопроцессора возвращается также вещественная неопределенность.

Результаты операций, приводящих к исключению

Операция	Результат
ошибка стека	вещественная неопределенность
операция с неподдерживаемым числом	вещественная определенность
операция с SNAN	QNAN
сравнение числа с NAN	C0 = C2 = C3 = 1
сложение бесконечностей с разным знаком или вычитание – с одним	вещественная неопределенность
умножение нуля на бесконечность	вещественная неопределенность
деление бесконечности на бесконечность или 0/0	вещественная неопределенность
команды FPREM и FPREM1, если делитель – 0 или делимое – бесконечность	вещественная неопределенность и C2 = 0
команды FCOS, FPTAN, FSIN, FSINCOS, когда аргументом является бесконечность	вещественная неопределенность и C2 = 0
корень или логарифм, если $x < 0$; $\log(x+1)$, если $x < -1$ (но FSQRT(–0)=0, FYL2X(–0)=∞)	вещественная неопределенность

команды FIST(P), когда регистр-источник пустой, содержит нечисло, бесконечность или превышает диапазон получателя	целочисленная неопределенность с кодом 10…0 (совпадает с минимальным числом используемого формата)
FBSTP, если регистр-источник пуст, содержит NAN, бесконечность или превышает 18 десятичных знаков	десятичная неопределенность с кодом 1111 1111 1111 1111 x…x здесь x – любое значение
FXCH, если хотя бы один из операндов пуст	пустой операнд заменяется вещественной неопределенностью и производится обмен

10.11.2. Деление на ноль

Исключение деления на ноль возникает в командах, которые явно или неявно выполняют деление. Например, это может быть команда FDIV. Факт возникновения этого исключения фиксируется флагом ZE (Zero Divide) регистра состояния сопроцессора SWR и при необходимости маскируется битом ZM регистра управления CWR. Маскированная реакция сопроцессора заключается в формировании результата в виде знаковой бесконечности.

10.11.3. Денормализация операнда

Исключение денормализованного операнда возникает, когда команда пытается выполнить операцию с денормализованным операндом. При этом устанавливается флаг DE (Denormalized Operand), который маскируется битом DM регистра управления CWR. Если это исключение замаскировано, то его возникновение приводит только к установке флага DE, после чего сопроцессор нормализует операнд и вычислительный процесс продолжается.

10.11.4. Переполнение и антипереполнение

Ситуации переполнения и антипереполнения возникают в случаях, когда порядок результата слишком велик или слишком мал для формата приемника. При возникновении этих исключений в регистре SWR устанавливаются флаги OE (Overflow) и UE (Underflow). Эти исключения маскируются битами OM и UM регистра управления CWR. Исключения могут возникнуть при работе арифметических команд и команд, преобразующих формат операндов, таких как FST.

Маскированная реакция для ситуации переполнения состоит в формировании граничных (максимальных или минимальных) значений, представимых в сопроцессоре, или специального численного значения в виде знаковой бесконечности (в зависимости от режима округления).

Маскированная реакция для ситуации антипереполнения: результат превращается в денормализованное число или нуль, регистрируется также исключение неточного результата.

10.11.5. Неточный результат

Исключение неточного результата возникает в случае, когда результат работы команды нельзя точно представить в формате приемника и его приходится округлять. Например, вычисление любой периодической дроби вроде 1/3 будет приводить к возникновению такого типа исключения.

Рассматриваемое исключение возникает довольно часто и показывает, что произошла некоторая (обычно приемлемая) потеря точности. Обычно он возникает в трансцендентных командах, что объясняется их природой.

В какую сторону произошло округление, можно судить по значению бита C1:

– если C1 = 0, то результат был усечен;

– если C1 = 1, то результат был округлен в большую сторону.

При возникновении этих исключений в регистре SWR устанавливаются флаг PE (Precision). Исключение маскируется битом PM регистра управления CWR.

Это исключение предусмотрено для приложений, рассчитанных только на точную арифметику. Большинству программистов о нем можно не заботиться.

10.11.6. Приоритет особых случаев

Микропроцессор обрабатывает исключения в соответствии с установленным старшинством. Старшинство означает, что отмечается исключение с высоким приоритетом и результат формируется в соответствии с требованиями этого исключения. Исключения с низким приоритетом не отмечаются даже при их появлении.

Принято следующее старшинство численных исключений:

1. Исключение недействительной операции, подразделяемое на:

– антипереполнение стека;

– переполнение стека;

– операнд в неподдерживаемом формате;

– операнд является сигнальным нечислом.

2. Операнд является тихим нечислом.

3. Любое другое, не упомянутое выше, исключение недействительной операции или деление на нуль.

4. Денормализованный операнд. Если это исключение замаскировано, выполнение команды продолжается и может возникнуть исключение с меньшим приоритетом.

5. Численные переполнение и антипереполнение. Может отмечаться также неточный результат.

6. Неточный результат.

Например, деление сигнального нечисла на нуль вызывает исключение недействительной операции (из-за нечисла), а не деления на нуль; маскированным результатом будет вещественная неопределенность, а не бесконечность.