Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»

Е. А. РУЖИЦКАЯ

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ASSEMBLER

Архитектура и программирование сопроцессора

Практическое пособие

для студентов специальностей
1–40 01 01 «Программное обеспечение
информационных технологий»,
1–40 04 10 «Информатика и технологии программирования»

Гомель
ГГУ им. Ф. Скорины
2016

УДК 004.431.4(076)
ББК 32.973.21я73
Р837

Рецензенты:
кандидат физико-математических наук Т. В. Тихоненко,
кандидат технических наук В. Д. Левчук

Рекомендовано к изданию научно-методическим советом
учреждения образования «Гомельский государственный
университет имени Франциска Скорины»

Ружицкая, Е. А.
Р837 Программирование на языке Assembler : архитектура и программирование сопроцессора : практическое пособие / Е. А. Ружицкая; М-во образования Республики Беларусь, Гомельский гос. ун-т им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2016. – 46 с.
ISBN 987-985-577-118-1

Практическое пособие предназначено для оказания помощи студентам в обладении машинно-ориентированным языком программирования Assembler. Излагается теоретический материал и дается практическое руководство по архитектуре, программированию и системе команд сопроцессора.

Адресовано студентам 1 курса специальностей 1–40 01 01 «Программное обеспечение информационных технологий», 1–40 04 10 «Информатика и технологии программирования».

УДК 004.431.4(076)
ББК 32.973.21я73

ISBN 987-985-577-118-1

Оглавление

Предисловие

Важной частью архитектуры микропроцессоров Intel является наличие устройства для обработки числовых данных в формате с плавающей точкой, называемого *математическим сопроцессором*. Архитектура компьютеров на базе микропроцессоров вначале опиралась исключительно на целочисленную арифметику. С ростом мощи компьютеров стали появляться устройства для обработки чисел с плавающей точкой. В архитектуре семейства микропроцессоров Intel 8086 устройство для обработки чисел с плавающей точкой появилось в составе компьютера на базе микропроцессора i8086/88 и получило название математический сопроцессор, или просто *сопроцессор*. Выбор такого названия был обусловлен тем, что, во-первых, это устройство было предназначено для расширения вычислительных возможностей основного процессора; 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 (необязательные в конкретной комплектации компьютера) устройства, сопроцессоры сохранялись вплоть до модели микропроцессора i386 и имели название i287 и i387. Начиная с модели i486, сопроцессор исполняется в одном корпусе с основным микропроцессором и, таким образом, является неотъемлемой частью компьютера.

Практическое пособие предназначено для оказания помощи студентам в обладении машинно-ориентированным языком программирования Assembler. Излагается теоретический материал и дается практическое руководство по архитектуре, программированию и системе команд сопроцессора.

Язык программирования Assembler изучается студентами 1 курса специальностей 1–40 01 01 «Программное обеспечение информационных технологий» в рамках дисциплины «Языки программирования» и 1–40 04 10 «Информатика и технологии программирования» в курсе «Программирование».

Тема 1. Архитектура сопроцессора

Сопроцессор дополняет возможности основного процессора и предназначен для выполнения следующих функций:

– поддержки арифметики с плавающей точкой;

– поддержки численных алгоритмов вычисления значений тригонометрических функций, логарифмов и т. п.;

– обработки десятичных чисел с точностью до 18 разрядов, что позволяет сопроцессору выполнять арифметические операции без округления над целыми десятичными числами со значениями до 10^{18};

– обработки вещественных чисел из диапазона $3{,}37 \cdot 10^{-4932}$ … $1.18 \cdot 10^{+4932}$.

С точки зрения программиста, сопроцессор представляет собой совокупность регистров, каждый из которых имеет свое функциональное назначение (рисунок 1.1). 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.

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.

Рисунок 1.1 – Программная модель сопроцессора

Регистровый стек сопроцессора организован по принципу кольца. Это означает, что среди всех регистров, составляющих стек, нет такого, который является вершиной стека. Все регистры стека с функциональной точки зрения абсолютно одинаковы и равноправны. В стеке есть вершина, которая является плавающей. Контроль текущей вершины осуществляется аппаратно с помощью трехбитового поля `top` регистра `swr` (рисунок 1.2). В поле `top` фиксируется номер регистра стека 0...7 (`r0,…,r7`), являющегося в данный момент текущей вершиной стека.

b	c3	top			c2	c1	c0	es	sf	pe	ue	oe	ze	de	ie
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Рисунок 1.2 – Регистр состояния swr

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.

Рассмотрим, каким образом «уживаются» между собой процессор и сопроцессор. Каждое из этих устройств имеет 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. Начиная с модели i486 сопроцессор и основной процессор производятся в одном корпусе и являются физически неделимыми, то есть архитектурно это по-прежнему два разных устройства, а аппаратно – одно.

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, как уже было отмечено, 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 в 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.

Например, 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 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 заканчивал работу над последней командой. В моделях микропроцессора, начиная с i486, подобная синхронизация выполняется командами `wait/fwait`, которые введены в алгоритм работы большинства команд сопроцессора.

Использование сопроцессора является совершенно прозрачным для программиста. В общем случае можно воспринимать сопроцессор как набор дополнительных регистров, для работы с которыми предназначены специальные команды.

1.1 Регистр состояния `swr`

Регистр `swr` отражает текущее состояние сопроцессора после выполнения последней команды. Структурно регистр `swr` (рисунок 1.2) состоит из:

– 6 флагов исключительных ситуаций;

– бита `sf` (Stack Fault) – ошибки 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 этот бит проанализирован, его нужно снова установить в ноль, вместе с битами `pe`, `ue` или `ie` (если они были установлены);

– бита `es` (Error Summary) – суммарной ошибки работы сопроцессора. Бит устанавливается в единицу, если возникает любая из шести перечисленных ниже исключительных ситуаций;

– четырех битов `c0–c3` (Condition Code) – кодов 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. Прерывания по месту их возникновения делятся на внешние и внутренние. Внутренние прерывания возникают в ходе работы текущей программы и делятся на синхронные (по команде `int`) и асинхронные, называемые *исключениями* или *особыми случаями*. Таким образом, *исключения* – это разновидность прерываний, с помощью которых процессор информирует программу о некоторых особенностях ее реального исполнения. Сопроцессор также обладает способностью возбуждения подобных прерываний при воз-

никновении 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 (рисунок 1.3). 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. Например, 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.

1.2 Регистр управления cwr

Регистр управления работой сопроцессора определяет особенности обработки численных данных (рисунок 1.3). Он состоит из:

– шести масок исключений;

– поля управления точностью pc (Precision Control);

– поля управления округлением rc (Roimding Control).

Шесть масок предназначены для маскирования исключительных ситуаций, возникновение которых фиксируется с помощью шести бит регистра swr. Если какие-то 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.

				rc		pc				p	u	o	z	d	i
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Рисунок 1.3 – Регистр управления сопроцессором cwr

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

– 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 округлением 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. Для того чтобы выяснить характер округления, введем обозначения:

– *m* – значение в st(0) или результат работы некоторой команды, который не может быть точно представлен и поэтому должен быть округлен;

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

Приведем значения поля rc и опишем соответствующий им характер округления:

– 00 – значение *m* округляется к ближайшему числу *a* или *b*;

– 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*. Используется для приведения значения к форме, которая может использоваться в операциях целочисленной арифметики.

1.3 Регистр тегов twr

Регистр тегов twr представляет собой совокупность двухбитовых полей. Каждое двухбитовое 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. Это значение в одном из двухбитовых полей регистра тегов не означает, что все биты соответствующего регистра стека должны быть обязательно нулевыми.

1.4 Форматы данных

Сопроцессор расширяет номенклатуру форматов данных, с которыми работает основной процессор. Сопроцессор специально разрабатывался для вычислений с плавающей точкой. Но сопроцессор может работать и с целыми числами, хотя и менее эффективно. Форматы данных, с которыми работает сопроцессор:

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

– упакованные целые десятичные (BCD) числа – максимальная длина 18 упакованных десятичных цифр (9 байт);

– вещественные числа в трех форматах – коротком (32 бита), длинном (64 бита) и расширенном (80 бит).

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

– денормализованные вещественные числа – это числа, меньшие минимального нормализованного числа для каждого вещественного формата, поддерживаемого сопроцессором;

– нуль;

– положительные и отрицательные значения бесконечность;

– нечисла;

– неопределенности и неподдерживаемые форматы.

В самом сопроцессоре числа в этих форматах имеют одинаковое внутреннее представление – в виде расширенного формата вещественного числа. Это один из форматов представления вещественных чисел, который точно соответствует формату регистров `r0…r7` стека сопроцессора.

Таким образом, даже если используются команды сопроцессора с целочисленными операндами, то после загрузки в сопроцессор операндов целого типа они автоматически преобразуются в формат расширенного вещественного числа.

1.4.1 Двоичные целые числа

Сопроцессор работает с тремя типами целых чисел:

– целое слово, 16 бит, –32 768…+32 767;

– короткое целое, 32 бита $-2\cdot10^9...+2\cdot10^9$;

– длинное целое 64 бита, $-9\cdot10^{18}...+9\cdot10^{18}$.

Сопроцессор поддерживает операции с целыми числами, но работа с ними осуществляется неэффективно. Причина в том, что обработка сопроцессором целочисленных данных будет замедлена из-за необходимости выполнения дополнительного преобразования целых чисел в их внутреннее представление в виде эквивалентного вещественного числа расширенного формата.

Рисунок 1.4 – Форматы целых чисел сопроцессора

В программе целые двоичные числа описываются с использованием директив `dw`, `dd` и `dq`. Например, целое число 5 может быть описано следующим образом:

```
ch_dw dw 5 представление в памяти: ch_dw=05 00
ch_dd dd 5 представление в памяти: ch_dw=05 00 00 00
ch_dq dq 5 представление в памяти: ch_dt=05 00 00 00 00 00 00 00
```

1.4.2 Упакованные целые десятичные (BCD) числа

Сопроцессор использует один формат упакованных десятичных чисел (рисунок 1.5). Для описания упакованного десятичного числа используется директива `dt`. Данная директива позволяет описать 20 цифр в упакованном десятичном числе (по две в каждом байте). Из-за того что максимальная длина упакованного десятичного числа в сопроцессоре составляет только 9 байт, в регистры `r0…r7` можно поместить только 18 упакованных десятичных цифр. Старший десятый байт игнорируется. Самый старший бит этого байта используется для хранения знака числа. Например, целое число 5365904 в формате упакованного десятичного числа может быть описано следующим образом:

```
ch_dt dt 5365904
;представление в памяти: ch_dt=04 59 36 05 00 00 00 00 00 00
```

Сопроцессор имеет для работы с упакованными десятичными числами всего две команды – сохранения и загрузки.

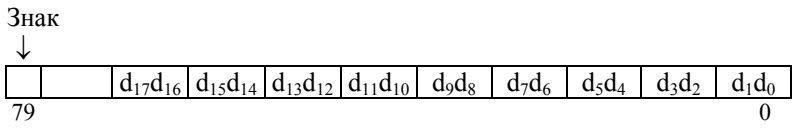

Рисунок 1.5 – Формат десятичного числа сопроцессора

1.4.3 Вещественные числа

Основной тип данных, с которыми работает сопроцессор, – вещественный. Данные этого типа описываются тремя форматами: коротким, длинным и расширенным (рисунок 1.6).

Рисунок 1.6 – Форматы вещественных чисел сопроцессора

Для представления вещественного числа используется формула:

$$A = (\pm M) \cdot N^{\pm(p)}, \qquad (1.1)$$

где 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 (по этой причине вещественные числа имеют еще название чисел с плавающей точкой, так как ее положение в разрядах мантиссы зависит от значения порядка).

Для удобства обработки в компьютере чисел с плавающей точкой, архитектурой компьютера на компоненты формулы (1.1) накладываются следующие ограничения:

– основание системы счисления $N = 2$;

– мантисса M должна быть представлена в *нормализованном* виде. Для архитектуры микропроцессора Intel нормализованным является число вида:

$$A = (-1)^s \cdot N^q \cdot M, \qquad (1.2)$$

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

p – порядок числа. Его значение аналогично значению порядка p в формуле (1.1).

В этой формуле знак имеют и порядок вещественного числа, и его мантисса. На рисунке 1.6 видно, что формат хранения вещественного числа в памяти имеет только поле для знака мантиссы. А где же хранится знак порядка? В сопроцессоре Intel на аппаратном уровне принято соглашение, что порядок p определяется в формате вещественного числа особым значением, называемым *характеристикой q*. Величина q связана с порядком p посредством формулы (1.3) и представляет собой некоторую константу. Условно назовем ее *фиксированным смещением*:

$$q = p + \text{фиксированное смещение}. \qquad (1.3)$$

Для каждого из трех возможных форматов вещественных чисел смещение q имеет разное, но фиксированное для конкретного формата значение, которое зависит от количества разрядов, отводимых под характеристику (таблица 1.1).

Таблица 1.1 – Формат вещественных чисел

Формат	Короткий	Длинный	Расширенный
Длина числа (биты)	32	64	80
Размерность мантиссы M	24	53	64
Диапазон значений	$10^{-38}...10^{38}$	$10^{-308}…10^{308}$	$10^{-4932}…10^{4932}$
Размерность характеристики q	8	11	15
Значение фиксированного смещения	+127 (7F) 0111 1111	+1023 (3FF) 0011 1111 1111	+16383 (3FFF) 0011 1111 1111 1111
Диапазон характеристик q	0...255	0...2047	0...32767
Диапазон порядков p	–126...+127	–1022...+1023	–16382...+16383

В таблице 1.1 показаны диапазоны значений характеристик q и соответствующих им истинных порядков p вещественных чисел. Нулевому порядку вещественного числа в коротком формате соответствует значение характеристики равное 127, которому в двоичном представлении соответствует значение 0111 1111. Отрицательному порядку p, например, –1, будет соответствовать характеристика $q = –1 + 127 = 126$.

В двоичном виде ей соответствует значение 0111 1110. Положительному порядку p, например, +1, будет соответствовать характеристика $q = 1 + 127 = 128$, в двоичном виде ей соответствует значение 1000 0000. То есть, все положительные порядки имеют в двоичном представлении характеристики старший бит равный единице, а отрицательные порядки – нет. Знак порядка спрятан в старшем бите характеристики.

Так как нормализованное вещественное число всегда имеет целую единичную часть (исключая представление перечисленных выше специальных численных значений), то при его представлении в памяти появляется возможность считать первый разряд вещественного числа единичным по умолчанию и учитывать его наличие только па аппаратном уровне. Это дает возможность увеличить диапазон представимых чисел, так как появляется лишний разряд, который можно использовать для представления мантиссы числа. Но это справедливо только для короткого и длинного форматов вещественных чисел. Расширенный формат, как внутренний формат представления числа любого типа в сопроцессоре, содержит целую единичную часть вещественного в явном виде.

Короткое вещественное число длиной в 32 разряда определяется директивой `dd`. При этом обязательным в записи числа является наличие десятичной точки, даже если оно 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 длинного и расширенного форматов представления вещественных чисел, определяемых директивами `dq` и `dt`.

Другой способ задания вещественного числа директивами `dd`, `dq` и `dt` – экспоненциальная форма с использованием символа «e».

Пример. Определим в программе вещественное число 45,56 в коротком формате:

```
dd 45.56
```

или

```
dd 45.56e0
```

или

```
dd 0.4556e2
```

В памяти это число будет выглядеть так:

```
71 3d 36 42
```

Так как в архитектуре Intel принят перевернутый порядок следования байт в памяти в соответствии с принципом «младший байт по младшему адресу», истинное представление числа 45,56 будет следующим: `42 36 3d 71`. В двоичном представлении в памяти число будет иметь вид, представленный на рисунке 1.7. На рисунке 1.7 видно, что старшая единица мантиссы при представлении в памяти отсутствует.

Переведем десятичную дробь 45,56 в эквивалентное двоичное представление:

$45{,}56_{10} = 101101{,}10001111010111000 01\ldots_2$.

Нормализуем число. Для этого переносим точку влево, до тех пор, пока в целой части числа не останется одна двоичная единица. Число переносов влево (или вправо, если десятичное число было меньше единицы) будет являться порядком числа. После перемещения точки получаем значение порядка $p = 5$. Соответственно, характеристика будет выглядеть так: $q = p + 127 = 5 + 127 = 132_{10} = 1000\ 0100_2$.

Сформированный результат в виде вещественного числа в коротком формате состоит из трех компонент:

– знака – 0;

– характеристики 1000 0100;

– мантиссы 0110 1100 0111 1010 1110 001…, содержащей целую чисть числа 45 без первой значащей единицы (01101) и дробную часть числа (100 0111 1010 1110 001…).

Рисунок 1.7 – Двоичное представление вещественного числа в коротком формате

Пример. Определим в программе вещественное число 45,56 в длинном формате:

```
dq 45.56
```

или

```
dq 45.56e0
```

В памяти это число будет выглядеть так:

```
47 e1 7a 14 ae c7 46 40
```

Перевернув его, получим истинное значение:

```
40 46 c7 ae 14 7a el 47
```

Для представления числа 45,56 в регистрах сопроцессора переведем это число в двоичную систему счисления:

$45{,}56_{10} = 101101{,}10001111010111000 01\ldots_2$.

Порядок числа $p = 5$.

Характеристика числа

$$q = p + 1023 = 5 + 1023 = 1028_{10} = 100\ 0000\ 0100_2.$$

Сформированный результат в виде вещественного числа в длинном формате состоит из трех компонент:

– знака – 0;

– характеристики 100 0000 0100;

– мантиссы 0110 1100 0111 1010 1110 001…, содержащей целую чисть числа 45 без первой значащей единицы (01101) и дробную часть числа (100 0111 1010 1110 001…).

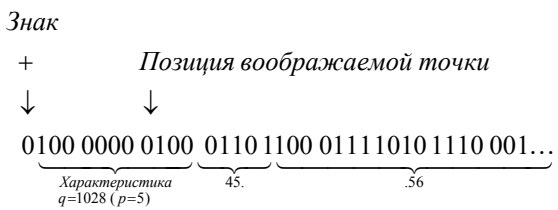

Рисунок 1.8 – Двоичное представление вещественного числа в длинном формате

Пример. Определим в программе вещественное число 45,56 в расширенном формате:

```
dt 45.56
```

В памяти это число будет выглядеть так:

```
71 3d 0a d7 a3 70 3d b6 04 40
```

Перевернув его, получим истинное значение в памяти:

```
40 04 b6 3d 70 a3 d7 0a 3d 71
```

Для представления числа 45,56 в регистрах сопроцессора переведем это число в двоичную систему счисления:

$45{,}56_{10} = 101101{,}1000111101011100001\ldots_2$.

Характеристика числа

$q = p + 16383 = 5 + 16383 = 16388_{10} = 100\ 0000\ 000\ 0100_2$.

Сформированный результат в виде вещественного числа в длинном формате состоит из трех компонент:

– знака – 0;

– характеристики 100 0000 0000 0100;

– мантиссы 1011 0110 0011 1101 0111 0001…, содержащей целую часть числа 45 с первой значащей единицей (101101) и дробную часть числа (100 0111 1010 1110 001…).

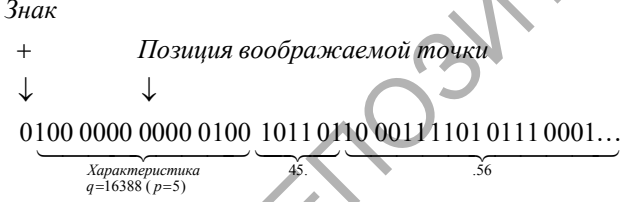

Рисунок 1.9 – Двоичное представление вещественного числа в расширенном формате

В мантиссе явно присутствует старшая единица, чего не было в коротком и длинном форматах представления вещественного числа.

1.4.4 Специальные численные значения

Несмотря на большой диапазон вещественных значений, представимых в регистрах стека сопроцессора, 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 бит, называемые *специальными численными значениями*. При необходимости программист может сам кодировать специальные численные значения. Это возможно потому, что вещественные числа, описанные директивой `dt`, соответствующие команды сопроцессора загружают без всяких преобразований.

Денормализованные вещественные числа – это числа, которые меньше минимального нормализованного числа для каждого вещественного формата. Например, для вещественного числа в расширенном формате диапазон представимых значений в сопроцессоре показан на рисунке 1.9.

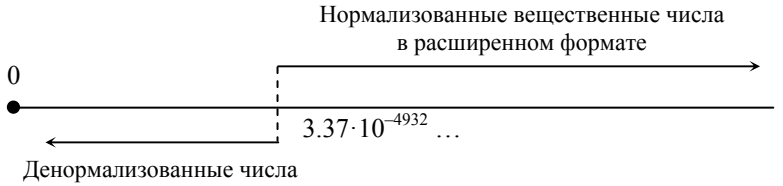

Рисунок 1.10 – Положение денормализованных вещественных чисел на числовой шкале

Как известно, сопроцессор хранит числа в нормализованном виде. По мере приближения чисел к нулю ему все труднее «вытягивать» их значения к нормализованному виду, то есть к такому виду, чтобы первой значащей цифрой мантиссы была единица. Размерность разрядной сетки, отведенной в форматах вещественных чисел сопроцессора, для представления характеристики не безгранична. Поэтому при определенных значениях числа в расширенном формате значение характеристики становится равным нулю (рисунок 1.10). Но на самом деле число отлично от нуля, так как это все же не настоящий численный нуль. Таким образом, между истинным нулем и минимально представимым нормализованным числом есть еще бесконечное количество очень маленьких чисел. Это и есть так называемые денормализованные числа. Они имеют нулевой порядок и ненулевую мантиссу. Диапазон представимых

в сопроцессоре денормализованных чисел не безграничен, так как количество разрядов мантиссы ограниченно (рисунок 1.11).

При формировании денормализованного значения в некотором регистре стека в соответствующем этому регистру теге регистра `twr` формируется специальное значение (10).

Минимальное положительное денормализованное число:

0	000000000000000	000000000000…001
79	78 64	63 0

Минимальное отрицательное денормализованное число:

1	000000000000000	000000000000…001
79	78 64	63 0

Максимальное положительное денормализованное число:

0	000000000000000	111111111111…111
79	78 64	63 0

Максимальное отрицательное денормализованное число:

1	000000000000000	111111111111…111
79	78 64	63 0

Рисунок 1.11 – Диапазон представимых в сопроцессоре денормализованных чисел

Нуль. Значение *нуля* 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 (рисунок 1.12). Если необходимо определить знак нуля, то используется команда `fxam`. В результате работы этой команды в бит `c1` регистра `swr` заносится знак операнда. При загрузке нуля в регистр стека в соответствующем теге регистра `twr` формируется специальное значение (01).

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

0	000000000000000	000000000000…000
79	78 64	63 0

1	000000000000000	000000000000…000
79	78 64	63 0

Рисунок 1.12 – Представление нуля в регистре стека сопроцессора

Значение бесконечность. Сопроцессор имеет средства для представления значения бесконечность. Это значение формируется с помощью специальных битовых значений. Формат регистра стека сопроцессора, содержащего значение бесконечность, приведен на рисунке 1.13. Значение бесконечность может иметь знак, при этом значения мантиссы и характеристики фиксированны. Именно в этом заключается отличие значения бесконечность от остальных специальных значений. Среди причин, приводящих к формированию значения бесконечность, можно выделить переполнение и деление на нуль. При формировании значения бесконечность в некотором регистре стека, в соответствующем теге регистра `twr` формируется специальное значение (10).

0	111111111111111	100000000000…000
79	78 64	63 0

1	111111111111111	100000000000…000
79	78 64	63 0

Рисунок 1.13 – Представление значения бесконечность в регистре стека сопроцессора

Нечисла. К *нечислам* относятся такие битовые последовательности в регистре стека сопроцессора, которые не совпадают ни с одним из рассмотренных выше форматов значений. Нечисло должно иметь единичную мантиссу и любую мантиссу, кроме 100...00, которая зарезервирована для значения бесконечность. Различают два типа нечисел:

– `SNAN` (Signaling Noil a Number) – сигнальные нечисла;

– `QNAN` (Quiet Non A Number) – спокойные (тихие) нечисла.

Сигнальное нечисло – битовое значение с единичным значением поля характеристики и мантиссы, первый бит которой, следующий за первым единичным значащим битом мантиссы, равен нулю (рисунок 1.14, *а*). Сопроцессор реагирует на появление этого числа в регистре стека возбуждением исключения недействительная операция. Программисты могут формировать эти числа в регистре стека сопроцессора преднамеренно, например, для того чтобы искусственно возбудить в нужной ситуации указанное исключение. Именно по этой причине данные числа называются сигнальными. Если снять маску у флага недействительная операция в регистре `cwr`, то будет вызван обработчик, который выполнит заданные программистом действия.

Спокойное нечисло – битовое значение с единичным значением полей характеристики и мантиссой, первые два бита которой равны единице (рисунок 1.14, *б*).

При формировании нечисла в некотором регистре стека в соответствующем теге регистра `twr` формируется специальное значение (10).

а)

x	111111111111111	10xxxxxxxxxx…xxx
79 78	64	63 0

б)

x	111111111111111	11xxxxxxxxxx…xxx
79 78	64	63 0

в)

1	111111111111111	110000000000…000
79 78	64	63 0

Рисунок 1.14 – Представление нечисел в регистре стека сопроцессора

Сопроцессор самостоятельно не формирует сигнальных чисел, но в качестве реакции па определенные исключения он может формировать спокойные нечисла, например, нечисло *вещественная неопределенность*. Его значение показано на рисунке 1.14, *в*. Вещественная неопределенность формируется как маскированная реакция сопроцессора на исключение недействительная операция. Другие спокойные нечисла могут формироваться после выполнения команд, в которых хотя бы один из операндов был спокойным нечислом. Это может породить цепную реакцию, которая приведет к ошибочному результату. Поэтому в процессе вычислений рекомендуется периодически контролировать результаты исполнения команд на предмет появления спокойных нечисел.

1.5 Практическое задание

Записать машинное представление вещественного числа в коротком, длинном и расширенном форматах в двоичной и шестнадцатеричной системах счисления.

Таблица 1.2 – Варианты заданий

Вариант	Число	Вариант	Число	Вариант	Число
1	36,25	11	46,5	21	56,75
2	37,5	12	47,75	22	57,25
3	38,75	13	48,25	23	58,5
4	39,25	14	49,5	24	59,75
5	40,5	15	50,75	25	60,25
6	41,75	16	51,25	26	62,5
7	42,25	17	52,5	27	62,75
8	43,5	18	53,75	28	63,25
9	44,75	19	54,25	29	64,5
10	45,25	20	55,5	30	65,75

Тема 2. Программная модель микропроцессора. Регистры

2.1 Система команд сопроцессора

Система команд сопроцессора включает около 80 машинных команд.

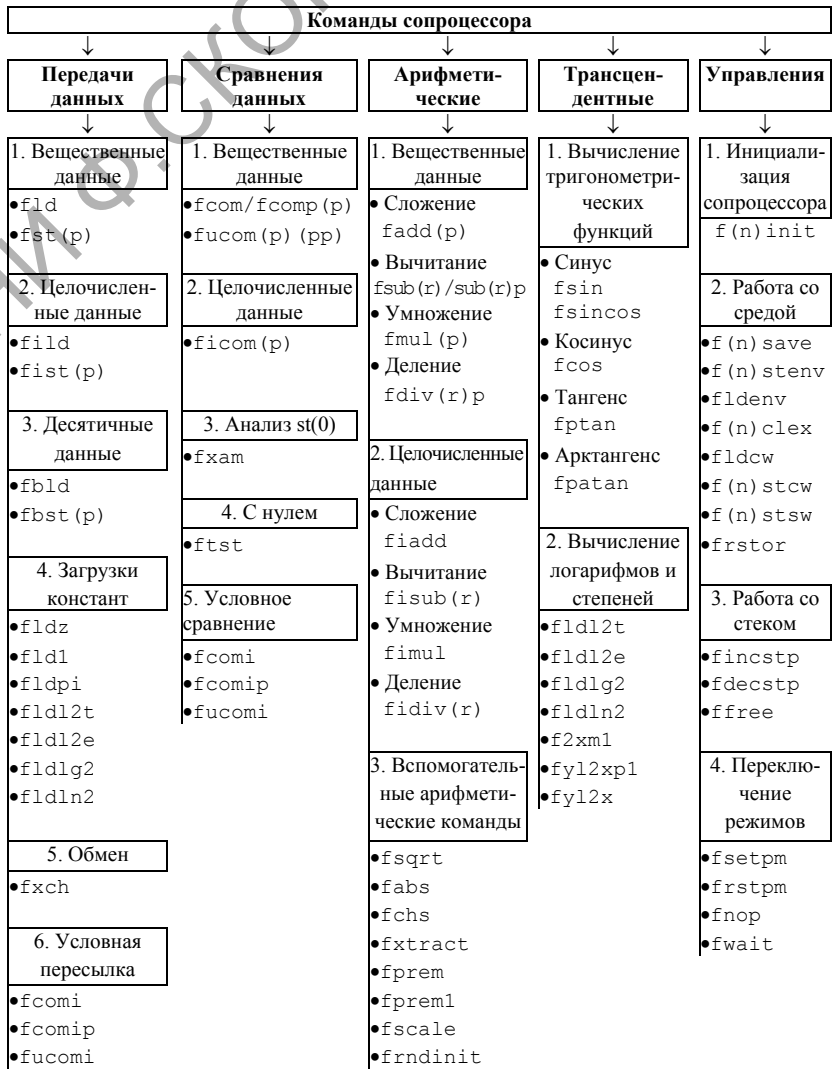

Рисунок 2.1 – Функциональная классификация команд сопроцессора

Мнемоническое обозначение команд сопроцессора характеризует особенности их работы:

1. Все мнемонические обозначения начинаются с символа `f` (float).

2. 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.

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

4. Последняя или предпоследняя буква `r` (reversed) в мнемоническом обозначении команды 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.

Методика написания программ для сопроцессора имеет свои особенности. Главная причина – в стековой организации сопроцессора.

При разработке программ необходимо учитывать:

– ограниченность глубины стека сопроцессора;

– несовпадение форматов операндов;

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

2.2 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. Главной функцией всех команд загрузки данных в сопроцессор является преобразование данных 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 3 группы:

1) команды передачи данных в вещественном формате;

2) команды передачи данных в целочисленном формате;

3) команды передачи данных в десятичном формате.

2.2.1 Команды передачи данных в вещественном формате

`fld источник` – загрузка вещественного числа из области памяти на вершину стека сопроцессора.

`fst приемник` – сохранение вещественного числа из вершины стека сопроцессора в память. Как следует из анализа мнемокода 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).

`fstp приемник` – сохранение вещественного числа из вершины стека сопроцессора в память. В отличие от предыдущей команды, в конце мнемонического обозначения данной команды присутствует символ `p`, что означает 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.

2.2.2 Команды передачи данных в целочисленном формате

`fild источник` – загрузка целого числа из памяти на вершину стека сопроцессора.

`fist приемник` – сохранение целого числа из вершины стека сопроцессора в память. Сохранение целого числа в памяти не сопровождается выталкиванием его из стека, то есть текущая вершина стека сопроцессора не изменяется.

`fistp приемник` – сохранение целого числа из вершины стека в память. Аналогично сказанному ранее о команде `fstp`, последним действием команды является выталкивание числа из стека с одновременным преобразованием его в целое значение.

2.2.3. Команды передачи данных в десятичном формате

`fbld источник` – загрузка десятичного числа из памяти в вершину стека сопроцессора.

`fbstp приемник` – сохранение десятичного числа из вершины стека сопроцессора в области памяти. Значение выталкивается из стека после преобразования его в формат десятичного числа.

Для десятичных чисел нет команды сохранения значения в памяти без выталкивания из стека.

2.2.4 Команда обмена вершины регистрового стека `st(0)` с любым другим регистром стека сопроцессора `st(i)`

`fxch st(i)`

Действие команд загрузки `fld`, `fild` и `fbld` можно сравнить с командой `push` основного процессора. Аналогично ей (`push` умень-

шает 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.

Для наблюдения за состоянием регистров сопроцессора используется окно Numeric processor (View – Numeric Processor).

2.2.5 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.

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 π;

fldl2t – загрузка двоичного логарифма десяти;

fldl2e – загрузка двоичного логарифма экспоненты (числа *e*);

fldlg2 – загрузка десятичного логарифма двойки;

fldln2 – загрузка натурального логарифма двойки.

2.3 Команды сравнения данных

Команды сравнения данных сравнивают значение числа в вершине стека и операнда, указанного в команде.

fcom [операнд_в_памяти] – команда без операндов сравнивает два значения: одно находится в регистре st(0), другое в регистре st(1). Если указан операнд [операнд_в_памяти], то сравнивается значение в регистре st(0) стека сопроцессора со значением в памяти.

fcomp операнд – команда сравнивает значение в вершине стека сопроцессора st(0) со значением операнда, который находится в регистре или в памяти. Последним действием команды является выталкивание значения из st(0).

fcompp операнд – команда аналогична по действию команде fcom без операндов, но последним ее действием является выталкивание из стека значений обоих регистров, st(0) и st(1).

ficom операнд_в_памяти – команда сравнивает значение в вершине стека сопроцессора st(0) с целым операндом в памяти. Длина целого операнда – 16 или 32 бита.

ficomp операнд – команда сравнивает значение в вершине стека сопроцессора st(0) с целым операндом в памяти. После сравнения и установки битов c3...c0 команда выталкивает значение из st(0). Длина целого операнда – 16 или 32 бита.

ftst – команда не имеет операндов и сравнивает значения в st(0) с нулем (значением 00).

Предыдущие команды сравнения работают корректно, если операнды в них являются целыми или вещественными числами. Когда один из операндов оказывается нечислом, то фиксируется исключение недействительной ситуации, а коды условия c3...c0 соответствуют исключительной ситуации несравнимых или неупорядоченных операндов. Само же действие сравнения не производится.

Процессор предоставляет три команды, позволяющие все же произвести сравнение таких операндов, но как вещественных чисел без учета их порядков.

fucom st(i) – команда сравнивает значения (без учета их порядков) в регистрах стека сопроцессора st(0) и st(i).

fucomp st(i) – команда сравнивает значения (без учета их порядков) в регистрах стека сопроцессора st(0) и st(i). Последним действием команды является выталкивание значения из вершины стека.

fucompp st(i) – команда сравнивает значения (без учета их порядков) в регистрах стека сопроцессора st(0) и st(i). Последние два действия команды – выталкивание значения из вершины стека.

В результате работы команд сравнения в регистре состояния устанавливаются следующие значения битов кода условия c3, c2, c0:

– если st(0) > операнда, то 000;

– если st(0) < операнда, то 001;

– если st(0) = операнду, то 100;

– если операнды неупорядочены, то 111.

Для того чтобы получить возможность реагировать на эти коды командами условного перехода основного процессора (они реагируют на флаги в eflags), нужно записать сформированные биты условия c3, c2, c0 в регистр eflags. В системе команд сопроцессора существует команда fstsw, которая позволяет запомнить слово состояния сопроцессора в регистре ax или ячейке памяти. Далее значения нужных битов извлекаются и анализируются командами основного процессора.

Например, запись старшего байта слова состояния, в котором находятся биты c0...c3, в младший байт регистра eflags/flags осуществляется командой sahf. Эта команда записывает содержимое ah в младший байт регистра eflags/flags. После этого бит c0 записывается на место флага cf, c2 – на место pf, c3 – на место zf. Бит

c1 выпадает из общего правила, так как в регистре флагов на месте соответствующего ему бита находится единица. Анализ этого бита нужно проводить с помощью логических команд основного процессора. Таким образом, нужно применять те команды условного перехода, которые анализируют состояние указанных флагов.

К группе команд сравнения данных логично отнести и команду fxam, которая анализирует операнд в вершине стека сопроцессора st(0) и формирует значение битов c0, c1, c2, c3 в регистре состояния сопроцессора swr. По состоянию этих битов можно судить:

– о знаке мантиссы – знаковый бит операнда в st(0) заносится в бит c0 регистра swr;

– корректности записи вещественного числа в st(0) – идентифицируются пустой регистр, корректное вещественное число, нечисло и неизвестный формат;

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

Пример. Программа разбиения массива вещественных чисел в формате двойного слова на два массива. В первый массив помещаются все элементы, которые больше или равны нулю, а во второй – меньше нуля.

```
;Исследование команд сравнения данных
.586p
masm
model use16 small
.stack 100h
;сегмент данных
.data
;исходный массив
mas dd -2.0, 45.7, -9.4, 7.3, 60.3, -58.44, 890e7,
-98746e3
mas_h_0dd 8 dup (0)
;массив значений больше либо равных 0
i_mas_h_0 dd 0 ;текущий индекс в mas_h_0
mas_l_0 dd 8 dup (0) ;массив значений меньших 0
i_mas_l_0 dd 0 ;текущий индекс в mas_l_0
.code
main proc ;начало процедуры main
    mov ax, @data
    mov ds, ax
    xor esi, esi
    mov cx, 8  ;счетчик циклов
    finit
   ;приведение сопроцессора в начальное состояние
    fldz ;загрузка нуля в st(0)
```



```
cycl:
    fcom mas[esi*4]
;сравнение нуля в st(0) с очередным
;элементом массива mas
    fstsw ax ;сохранение swr в регистре ax
    sahf ;запись swr->ax-> регистр флагов
    jp error ;переход по "плохому" операнду
             ;в команде fcom
    jc hi_0 ;переход, если mas[esi*4]>= 0
            ;(mas[esi*4]>=st(0))
;пересылка операнда mas[esi*4]
;меньшего 0 в массив mas_l_0
    mov eax, mas[esi*4]
    mov edi, i_mas_l_0
    mov mas_l_0[edi*4], eax
    inc i_mas_l_0
    jmp cycl_bst
hi_0:
;пересылка операнда mas[esi*4] большего
;или равного 0 в массив mas_h_0
    mov eax, mas[esi*4]
    mov edi, i_mas_h_0
    mov mas_h_0[edi*4], eax
    inc i_mas_h_0
cycl_bst:
    inc si
    loop cycl
error:
;здесь можно вывести сообщение об ошибке
;в задании операндов
exit:
    mov ax, 4c00h
    int 21h
main endp
end main
```

2.4 Арифметические команды

Команды сопроцессора, входящие в группу арифметических команд, реализуют четыре основные арифметические операции – сложение, вычитание, умножение и деление.

С точки зрения типов операндов арифметические команды сопроцессора можно разделить на команды, работающие с вещественными и целыми числами.

2.4.1 Целочисленные арифметические команды

Целочисленные арифметические команды предназначены для работы на тех участках вычислительных алгоритмов, где в качестве исходных данных используются целые числа в памяти в форматах слово и короткое слово, имеющие размерность 16 и 32 бита.

`fiadd источник` – команда складывает значения `st(0)` и целочисленного источника, в качестве которого выступает 16- или 32-разрядный операнд в памяти. Результат сложения запоминается в регистре стека сопроцессора `st(0)` (`st(0)` = `st(0)` + *И*).

`fisub источник` – команда вычитает значение целочисленного источника из `st(0)`. Результат вычитания запоминается в регистре стека сопроцессора `st(0)`. В качестве источника выступает 16- или 32-разрядный целочисленный операнд в памяти.

`fimul источник` – команда умножает значение целочисленного источника на содержимое `st(0)`. Результат умножения запоминается в регистре стека сопроцессора `st(0)`. В качестве источника выступает 16- или 32-разрядный целочисленный операнд в памяти.

`fidiv источник` – команда делит содержимое `st(0)` на значение целочисленного источника. Результат деления запоминается в регистре стека сопроцессора `st(0)`. В качестве источника выступает 16- или 32-разрядный целочисленный операнд в памяти.

Для команд, реализующих арифметические действия деления и вычитания, важен порядок расположения операндов. По этой причине система команд сопроцессора содержит соответствующие реверсивные команды, повышающие удобство программирования вычислительных алгоритмов. Чтобы отличить эти команды от обычных команд деления и вычитания, их мнемокоды оканчиваются символом `r`.

`fisubr источник` – команда вычитает значение `st(0)` из целочисленного источника. Результат вычитания запоминается в регистре стека сопроцессора `st(0)`. В качестве источника выступает 16- или 32-разрядный целочисленный операнд в памяти.

`fidivr источник` – команда делит значение целочисленного источника на содержимое `st(0)`. Результат деления запоминается в регистре стека сопроцессора `st(0)`. В качестве источника выступает 16- или 32-разрядный целочисленный операнд в памяти.

Пример программы вычисления значения выражения:

$$u = \begin{cases} \dfrac{x+y}{a}, & a \neq 0; \\ x+y, & a = 0. \end{cases}$$

Все переменные *x*, *y* и *a* целого типа в формате слова. Результат необходимо сохранить в ячейке памяти в формате десятичного числа.


```
;Исследование целочисленных арифметических команд
.586p
masm
model use16 small
.stack 100h
.data
;сегмент данных
;исходные данные
a dw 0
x dw 8
y dw 4
u dt 0
;сегмент кода
.code
main proc
    mov ax, @data
    mov ds, ax
    finit
;приведение сопроцессора в начальное состояние
    fild a ;загрузка значение a в st(0)
    fxam ;определяем тип a
    fstsw ax ;сохранение swr в регистре ax
    sahf ;запись swr->ax-> регистр флагов
    jc no_null
    jp no_null
    jnz no_null
;вычисление формулы u=x+y:
    fild x
    fiadd y
    fbstp u
    jmp exit
no_null:
;вычисление формулы u=(x-y)/a:
    fild x
    fisub y
    fidiv a
    fbstp u
exit:
    mov ax, 4c00h
    int 21h
main endp
end main
```

2.4.2 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, 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.

Команды сложения

`fadd` – команда складывает значения в `st(0)` и `st(1)`. Результат сложения запоминается в регистре стека сопроцессора `st(0)`.

`fadd источник` – команда складывает значения `st(0)` и источника, представляющего адрес ячейки памяти. Результат сложения запоминается в регистре стека сопроцессора `st(0)`.

`fadd st(i),st` – команда складывает значение в регистре стека сопроцессора `st(i)` со значением в вершине стека `st(0)`. Результат сложения запоминается в регистре `st(i)`.

`faddp st(i),st` – команда производит сложение вещественных операндов аналогично команде `fadd st(i),st`, однако последним действием команды является выталкивание значения из вершины стека сопроцессора `st(0)`. Результат сложения остается в регистре `st(i-1)`.

Команды вычитания

`fsub` – команда вычитает значение в `st(1)` из значения в `st(0)`. Результат вычитания запоминается в регистре стека сопроцессора `st(0)`.

`fsub источник` – команда вычитает значение источника из значения в `st(0)`. Источник представляет адрес ячейки памяти, содержащей допустимое вещественное число. Результат сложения запоминается в регистре стека сопроцессора `st(0)`.

`fsub st(i),st` – команда вычитает значение в вершине стека `st(0)` из значения в регистре стека сопроцессора `st(i)`. Результат вычитания запоминается в регистре стека сопроцессора `st(i)`.

`fsubp st(i),st` – команда вычитает вещественные операнды аналогично команде `fsub st(i),st`. Последним действием команды является выталкивание значения из вершины стека сопроцессора `st(0)`. Результат вычитания остается в регистре `st(i-1)`.

`fsubr st(i),st` – команда вычитает значение в вершине стека `st(0)` из значения в регистре стека сопроцессора `st(i)`. Результат вычитания запоминается в вершине стека сопроцессора – регистре `st(0)`.

`fsubrp st(i),st` – команда производит вычитание подобно команде `fsubr st(i),st`. Последним действием команды является выталкивание значения из вершины стека сопроцессора `st(0)`. Результат вычитания остается в регистре `st(i-1)`.

Команды умножения вещественных операндов. Операнды располагаются исключительно в стеке сопроцессора.

`fmul` – команда не имеет операндов. Умножает значения в `st(0)` на содержимое в `st(1)`. Результат умножения запоминается в регистре стека сопроцессора `st(0)`.

`fmul st(i)` – команда умножает значение в `st(0)` на содержимое регистра стека `st(i)`. Результат умножения запоминается в регистре стека сопроцессора `st(0)`.

`fmul st(i),st` – команда умножает значения в `st(0)` на содержимое произвольного регистра стека `st(i)`. Результат умножения запоминается в регистре стека сопроцессора `st(i)`.

`fmulp st(i),st` – команда производит умножение подобно команде `fmul st(i),st`. Последним действием команды является выталкивание значения из вершины стека сопроцессора `st(0)`. Результат умножения остается в регистре `st(i-1)`.

Команды деления вещественных данных. Операнды этих команд располагаются в стеке сопроцессора:

`fdiv` – команда (без операндов) делит содержимого регистра `st(0)` на значение регистра сопроцессора `st(1)`. Результат деления запоминается в регистре стека сопроцессора `st(0)`.

`fdiv st(i)` – команда делит содержимое регистра `st(0)` на содержимое регистра сопроцессора `st(i)`. Результат деления запоминается в регистре стека сопроцессора `st(0)`.

`fdiv st(i),st` – команда производит деление аналогично команде `fdiv st(i)`, но результат деления запоминается в регистре стека сопроцессора `st(i)`.

`fdivp st(i),st` – команда производит деление аналогично команде `fdiv st(i),st`. Последним действием команды является выталкивание значения из вершины стека сопроцессора `st(0)`. Результат деления остается в регистре `st(i-1)`.

`fdivr st(i),st` – команда делит содержимое регистра `st(i)` на содержимое вершины регистра сопроцессора `st(0)`. Результат деления запоминается в регистре стека сопроцессора `st(0)`.

`fdivrp st(i),st` – команда делит содержимое регистра `st(i)` на содержимое вершины регистра сопроцессора `st(0)`. Результат деления запоминается в регистре стека сопроцессора `st(i)`, после чего производится выталкивание содержимого `st(0)` из стека. Результат деления остается в регистре `st(i-1)`.

2.4.3 Дополнительные арифметические команды

Далее перечислены команды, входящие в группу дополнительных арифметических команд:

`fsqrt` – вычисление квадратного корня из значения, находящегося в вершине стека сопроцессора – регистре `st(0)`. Команда не имеет операндов. Результат вычисления помещается в регистр `st(0)`.

`fabs` – вычисление модуля значения, находящегося в вершине стека сопроцессора – регистре `st(0)`. Команда не имеет операндов. Результат вычисления помещается в регистр `st(0)`.

`fchs` – изменение знака значения, находящегося в вершине стека сопроцессора – регистре `st(0)`. Команда не имеет операндов. Результат вычисления помещается обратно в регистр `st(0)`. Отличие команды `fchs` от команды `fabs` в том, что команда `fchs` только инвертирует знаковый разряд значения в регистре `st(0)`, не меняя значения остальных битов. Команда вычисления модуля `fabs` при наличии отрицательного значения в регистре `st(0)`, наряду с инвертированием знакового разряда, выполняет изменение остальных битов значения таким образом, чтобы в `st(0)` получилось соответствующее положительное число.

`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, и соответствует величине *p* формулы (1.3).

2.4.4 Команда масштабирования

`fscale` – эта команда 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.

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 последняя команда в группе дополнительных команд – команда `frndint`. Эта команда 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` управляющего регистра сопроцессора. С помощью двух команд `fstcwr` и `fldcwr` можно изменить режим округления. Эти команды, соответственно, записывают в память содержимое управляющего регистра сопроцессора и восстанавливают его обратно. Таким образом, пока содержимое этого регистра находится в памяти, можно установить необходимое значение поля `rc`.

2.5 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.

Команды трансцендентных функций:

`fcos` – команда вычисляет косинус угла, находящийся в вершине стека сопроцессора – регистре `st(0)`. Команда не имеет операндов. Результат возвращается в регистр `st(0)`.

`fsin` – команда вычисляет синус угла, находящийся в вершине стека сопроцессора – регистре `st(0)`. Команда не имеет операндов. Результат возвращается в регистр `st(0)`.

`fsincos` – команда вычисляет синус и косинус угла, находящиеся 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(0)`, а косинус – в `st(1)`.

`fptan` – команда вычисляет *частичный* тангенс угла, находящийся в вершине стека сопроцессора – регистре `st(0)`. Команда не имеет операндов. Результат возвращается в регистрах `st(0)` и `st(1)`.

`fpratan` – команда вычисляет *частичный* арктангенс угла, находящийся в вершине стека сопроцессора – регистре `st(0)`. Команда не имеет операндов. Результат возвращается в регистрах `st(0)` и `st(1)` (угол от 0 до $\pi / 4$).

Отметим еще две команды сопроцессора: `fprem` и `fprem1`.

Команда `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. Физически работа команды заключается в реализации хорошо известного всем действия: деления 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)`.

Важно отметить, то команда `fprem` не соответствует последнему стандарту IEEE-754 на вычисления с плавающей точкой. По этой причине в систему команд сопроцессора 1387 была 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`.

`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 диапазоне $-1 \le x \le 1$. Результат y замещает x в регистр `st(0)`.

`fyl2x` – команда вычисления значения функции $z = y \log_2(x)$. Исходное значение x размещается в вершине стека сопроцессора, а исходное значение y – в регистре `st(1)`. Значение x должно лежать в

диапазоне $0 < x < +\infty$, а значение y – в диапазоне $-\infty < y < +\infty$. Перед тем как осуществить запись результата z в вершину стека, команда `fyl2x` выталкивает значения x и y из стека и только после этого производит запись z в регистр `st(0)`.

Команда `fyl2xp1` – команда вычисления значения функции $z = y \log_2(x+1)$. 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 $0 < |x| < 1 - \frac{1}{\sqrt{2}}$, а значение y – в диапазоне $-\infty < y < +\infty$. Перед тем, как записать результат z в вершину стека – регистр `st(0)`, команда `fyl2xp1` выталкивает из стека значения x и y.

2.6 Команды управления сопроцессором

Эта 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, возникающих 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), различающиеся 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`.

`wait/fwait` – команда ожидания. Она предназначена для синхронизации работы процессора и сопроцессора.

Команда инициализации сопроцессора `finit/fninit`. Она инициализирует управляющие регистры сопроцессора определенными значениями.

Следующие две команды работают с регистром состояния `swr`.

`fstsw/fnstsw ax` – команда сохранения содержимого регистра состояния `swr` в регистре `ax`. Эту команду целесообразно использовать для подготовки к условным переходам по описанной при рассмотрении команд сравнения схеме.

`fstsw/fnstsw приемник` – команда сохранения содержимого регистра состояния `swr` в ячейке памяти. От рассмотренной ранее команда отличается типом операнда – теперь это ячейка памяти размером два байта (в соответствии с размерностью регистра `swr`).

Следующие две команды, работающие с информацией в регистре управления `cwr`, поддерживают действие записи и чтения содержимого этого регистра.

`fstcw/fnstcw приемник` – команда сохранения содержимого регистра управления `cwr` в ячейке памяти размером два байта. Эту команду целесообразно использовать для анализа полей маскирования исключений, управления точностью и округления. Следует заметить, что операндом не является регистр `ax`, в отличие от команды `fstsw/fnstsw`.

`fldcw источник` – команда загрузки значения ячейки памяти размером 16 битов в регистр управления `cwr`. Эта команда выполняет действие, противоположное действию команды `fstcw/fnstcw`. Команду `fldcw` целесообразно использовать для задания или изменения режима работы сопроцессора.

Команда без операндов `fclex/fnclex` позволяет сбросить флаги исключений в регистре состояния `swr` сопроцессора.

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.

`fdecstp` – команда уменьшения указателя стека (поле `top`) в регистре `swr`. Команда не имеет операндов.

`ffree st(i)` – команда помечает любой регистр стека сопроцессора как пустой. Это команда освобождения регистра стека `st(i)`.

В группе команд управления можно выделить подгруппу команд, работающих с так называемой средой сопроцессора. *Среда сопроцессора* – это совокупность регистров сопроцессора и их значений. Среда сопроцессора может быть *частичной* или *полной.* О какой именно среде идет речь, определяется одной из рассмотренных далее команд:

`fsave/fnsave приемник` – команда сохранения полного состояния среды сопроцессора в память, адрес которой указан операндом приемник.

`frstor источник` – команда 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`.

Пример вычисления выражения $y = a + bc - d/(a+b)$ с использованием целочисленных арифметических команд.

```
;Вычисление выражения y=a+b*c-d/(a+b)
;целочисленные арифметические команды сопроцессора
masm
model use16 small
.stack 100h
;сегмент данных
.data
a dw 2
b dw 3
c dw 4
d dw 5
z dw ?
y dw ?
;сегмент кода
.code
main proc
   mov ax, @data
   mov ds, ax
   finit
   ;приведение сопроцессора в начальное состояние
   fild b   ;загрузка значение b в st(0)
   fimul c  ;st(0)=bc
   fiadd a  ;st(0)=bc+a
   fistp z  ;z=bc+a
   fild  a  ;загрузка значение a в st(0)
   fiadd b  ;st(0)=b+a
   fidivr d ;st(0)=d/(b+a)
   fisubr z ;z=st(0)-z
   fistp y  ;результат в y
   mov ax,4c00h
   int 21h
main endp
end main
```

Пример вычисления выражения $y = a + bc - d/(a+b)$ с использованием вещественных арифметических команд.

```
;Вычисление выражения y=a+b*c-d/(a+b).
;Вещественные арифметические команды сопроцессора.
masm
model use16 small
.stack 100h
.data ;сегмент данных
a dd 2.5
b dd 3.5
c dd 4.5
d dd 5.5
z dd ?
y dd ?
.code
main proc
   mov ax,@data
   mov ds, ax
   finit
   ;приведение сопроцессора в начальное состояние
   fld b     ;загрузка значение b в st(0)
   fld c     ;загрузка значение c в st(0) st(1)=b
   fmul      ;st(0)=bc
   fadd a    ;st(0)=bc+a
   fstp z    ;z=bc+a
   fld a     ;загрузка значение a в st(0)
   fadd b    ;st(0)=b+a
   fld d     ;st(0)=d st(1)=b+a
   fdiv      ;st(0)=d/(b+a)
   fld z     ;st(0)=a+bc st(1)=d/(b+a)
   fxchst(1) ;st(0) и st(1) меняем местами
   fsub      ;st(0)=st(0)-st(1)
   fstp y    ;результат в y
   mov ax,4c00h
   int 21h
main endp
end main
```

Пример вычисления выражения $y = \begin{cases} ab, & a+b<5; \\ 2a-b, & a+b>5; \\ a/b, & a+b=5; \end{cases}$ с использованием целочисленных арифметических команд. Программа написана с использованием внутренних процедур (каждая «веточка» вычисляется в отдельной процедуре).

При вычислении значения этой функции необходимо анализировать условие. В таблице 2.1 представлены значения флагов и команды условного перехода, которые используются для проверки условий в регистрах сопроцессора.

Таблица 2.1 – Значения флагов и используемые команды условного перехода при работе с регистрами сопроцессора

Условия	c3	c2	c0	Команда условного перехода
st(0) > операнда	0	0	0	jnc
st(0) < операнда	0	0	1	jc
st(0) = операнда	1	0	0	jz
Неупорядоченная пара	1	1	1	
Флаги	zf	pf	cf	

Команды условного перехода:

```
jc metka      ;переход, если cf=1
jnc metka     ;переход, если cf=0
jp metka      ;переход, если pf=1
jnp metka     ;переход, если pf=0
jz metka      ;переход, если zf=1
jnz metka     ;переход, если zf=0
```

```
;Вычисление выражения y=a·b,    если a+b<5,
;                      y=2·a-b, если a+b>5,
;                      y=a/b,   если a+b=5.
;Целочисленные арифметические команды сопроцессора.
masm
model use16 small
.486
.stack 100h
.data ;сегмент данных
a dw 4
b dw 2
y dw ?
five dw 5
two  dw 2
.code
main proc
   mov ax,@data
   mov ds,ax
   finit
```

```
   ;приведение сопроцессора в начальное состояние
   fild a   ;загрузка значение a в st(0)
   fiadd b  ;st(0)=a+b
   ficomp five ;сравниваем st(0) с 5
           ;и одновременно сбрасываем регистр st(0)
   fstsw ax ;сохранение swr в регистре ax
   sahf     ;запись swr->ax-> регистр флагов
   jc met1  ;если a+b<5 переход на метку1
   jz met2  ;если a+b=5 переход на метку2
   call p2  ;вычисляем значение при a+b>5
   jmp exit
met1: call p1  ;вычисляем значение при a+b<5
   jmp exit
met2: call p3  ;вычисляем значение при a+b=5
exit: mov ax,4c00h
   int 21h
main endp
p1 proc     ;вычисляем значение при a+b<5
   fild a   ;st(0)=a
   fimul b  ;st(0)=a*b
   fist y
   ret
p1 endp
p2 proc      ;вычисляем значение при a+b>5
   fild a    ;st(0)=a
   fimul two ;st(0)=2*a
   fisub b   ;st(0)=2*a-b
   fist y    ;y=2*a-b
   ret
p2 endp
p3 proc ;вычисляем значение при a+b=5
   fild a   ;st(0)=a
   fidiv b  ;st(0)=a/b
   fist y
   ret
p3 endp
end main
```

Пример вычисления выражения $y=\begin{cases} ab, & a+b<5; \\ 2a-b, & a+b>5; \\ a/b, & a+b=5; \end{cases}$ с использованием вещественных арифметических команд. Программу написана с использованием внутренних процедур (каждая «веточка» вычисляется в отдельной процедуре).


```
;Вычисление выражения y=a*b,   если a+b<5,
;                      y=2*a-b, если a+b>5,
;                      y=a/b,   если a+b=5.
;Вещественные арифметические команды сопроцессора.
masm
model use16 small
.486
.stack 100h
.data ;сегмент данных
a dd 4.5
b dd 2.5
y dd ?
five dd 5.0
two  dd 2.0
.code
main proc
   mov ax, @data
   mov ds, ax
   finit
;приведение сопроцессора в начальное состояние
   fld a  ;загрузка значение a в st(0)
   fadd b ;st(0)=a+b
   fcomp five
   ;сравниваем st(0) с 5 и одновременно сбрасываем
   ;регистр st(0)
   fstsw ax ;сохранение swr в регистре ax
   sahf     ;запись swr->ax-> регистр флагов
   jc met1  ;если a+b<5 переход на метку1
   jz met2  ;если a+b=5 переход на метку2
   call p2  ;вычисляем значение при a+b>5
   jmp exit
met1: call p1 ;вычисляем значение при a+b<5
   jmp exit
met2: call p3 ;вычисляем значение при a+b=5
exit: mov ax, 4c00h
   int 21h
main endp
p1 proc    ;вычисляем значение при a+b<5
   fld a   ;st(0)=a
   fmul b  ;st(0)=a*b
   fst y
   ret
p1 endp
p2 proc    ;вычисляем значение при a+b>5
   fld a   ;st(0)=a
```

```
   fmul two  ;st(0)=2*a
   fsub b    ;st(0)=2*a-b
   fst y     ;y=2*a-b
   ret
p2 endp
p3 proc    ;вычисляем значение при a+b=5
   fld a   ;st(0)=a
   fdiv b  ;st(0)=a/b
   fst y
   ret
p3 endp
end main
```

2.7 Практическое задание

Составить и отладить программу на языке ассемблера для вычисления значения функции, используя регистры сопроцессора. Написать 2 варианта программы:

– с использованием целочисленных команд сопроцессора;

– с использованием вещественных команд сопроцессора.

Программу оформить с использованием внутренних процедур (каждая «веточка» вычисляется в отдельной процедуре). Программу протестировать по всем условиям. Варианты заданий даны в таблице 2.2.

Таблица 2.2 – Варианты заданий по системе команд сопроцессора

Вариант 1 $y=\begin{cases}6xy-4y, & x+y>9;\\ \dfrac{2xy+3+x}{x^2+3y^2+1}, & x+y<-1;\\ 3x^2-2y+6, & -1\le x+y\le 9.\end{cases}$	Вариант 2 $y=\begin{cases}4xy+5, & x+y>10;\\ \dfrac{3xy+2y}{x^2+y^2+1}, & x+y<-2;\\ 6x-2y^2+1, & -2\le x+y\le 1.\end{cases}$
Вариант 3 $y=\begin{cases}4xy+4x, & xy>8;\\ \dfrac{2xy+3}{x^2+2y^2-1}, & xy<-12;\\ 3x^2-2y^2+1, & -12\le xy\le 8.\end{cases}$	Вариант 4 $y=\begin{cases}2xy+3, & x-y>9;\\ \dfrac{3y^2}{x^2+2y+1}, & x-y<-6;\\ 2x^2-2y+1, & -6\le x-y\le 9.\end{cases}$
Вариант 5 $y=\begin{cases}3x+2y^2-5, & xy>8;\\ \dfrac{xy+7}{2x^2+4}, & xy<-12;\\ 4x^2-3y^2+4, & -12\le xy\le 8.\end{cases}$	Вариант 6 $y=\begin{cases}5x+2xy+1, & x+y>9;\\ \dfrac{2xy+3}{y^2+4}, & x+y<-5;\\ 3x^2-2y^2+1, & -5\le x+y\le 9.\end{cases}$

Вариант 7 $y=\begin{cases}3x+2y^2-1, & x+y>8;\\ \dfrac{2xy+3}{y^2+2x+1}, & x+y<-5;\\ 2x^2-3y^2+2, & -5\le x+y\le 8.\end{cases}$	Вариант 8 $y=\begin{cases}3x+4y+3, & x+y>7;\\ \dfrac{3y-2x^2}{y^2+4}, & x+y<-5;\\ 3x^2-4y^2-3, & -5\le x+y\le 7.\end{cases}$
Вариант 9 $y=\begin{cases}\dfrac{x^2+2y}{x^2+y^2+1}, & x-y>15;\\ \dfrac{y^2-x}{x^2+y^2-1}, & x-y<-5;\\ x^2+2y^2+4, & -5\le x-y\le 15.\end{cases}$	Вариант 10 $y=\begin{cases}\dfrac{x^2+3y}{x^2+2y^2+1}, & x+y>6;\\ \dfrac{3y^2-1}{2x^2+y^2+1}, & x+y<-5;\\ 2x^2+3y^2-4, & -5\le x+y\le 6.\end{cases}$
Вариант 11 $y=\begin{cases}\dfrac{y+2x}{x^2+3y^2+1}, & x+y>11;\\ \dfrac{4xy}{x^2+2y^2+1}, & x+y<-4;\\ 5x^2+2y^2, & -4\le x+y\le 11.\end{cases}$	Вариант 12 $y=\begin{cases}\dfrac{x^2-4y+5}{x^2+2}, & x-y>2;\\ \dfrac{2x+y}{3x^2+4}, & x-y<-4;\\ x^2-2y+3, & -4\le x-y\le 2.\end{cases}$
Вариант 13 $y=\begin{cases}\dfrac{2x+y+4}{xy^2+5}, & x-y>8;\\ 2+3x^2+y, & x-y<-2;\\ 5x^2-2y, & -2\le x-y\le 8.\end{cases}$	Вариант 14 $y=\begin{cases}\dfrac{1+x}{5x^2-y^2+1}, & x-y<-4;\\ 3x+5y^2, & x-y>6;\\ x^3-2y, & -4\le x-y\le 6.\end{cases}$
Вариант 15 $y=\begin{cases}5xy-3, & x+y>8;\\ \dfrac{4xy^2}{x^2+1}, & x+y<-5;\\ 2x^2-4y^2+3, & -5\le x+y\le 8.\end{cases}$	Вариант 16 $y=\begin{cases}x^2-2y, & x+y\le -3;\\ \dfrac{xy+5}{x^2+y^2+1}, & -3<x+y<5;\\ 2xy-3y^2, & x+y\ge 5.\end{cases}$
Вариант 17 $y=\begin{cases}4x+y-3, & x+y>2;\\ \dfrac{3xy+2y}{x^2+2y^2+1}, & x+y<-10;\\ 5x-3y^2+7, & -10\le x+y\le 2.\end{cases}$	Вариант 18 $y=\begin{cases}5xy+4y, & x+y>9;\\ \dfrac{xy-5}{x^2+2y^2-1}, & x+y<-1;\\ 3x^2-2y+6x, & -1\le x+y\le 9.\end{cases}$

Вариант 19 $y=\begin{cases} xy+5x, & xy>9; \\ \dfrac{2xy-3}{x^2+2y^2+2}, & xy<-1; \\ 4x-y^2+8, & -1\le xy\le 9. \end{cases}$	Вариант 20 $y=\begin{cases} 7xy-4, & xy>6; \\ \dfrac{xy+1}{x^2+y^2+3}, & xy<-10; \\ 3x^2-2y^2-1, & -10\le xy\le 6. \end{cases}$
Вариант 21 $y=\begin{cases} 3xy-1+y, & x+y>7; \\ \dfrac{x+y}{x^2+4}, & x+y<-10; \\ 3x^2-2y^2+1, & -10\le x+y\le 7. \end{cases}$	Вариант 22 $y=\begin{cases} 5xy-x^2, & x+y>9; \\ \dfrac{4xy}{y^2+1}, & x+y<-4; \\ 3x^2-y^2+4, & -4\le x+y\le 9. \end{cases}$
Вариант 23 $y=\begin{cases} 5xy-y+5, & xy>9; \\ \dfrac{2xy-1}{y^2+1}, & xy<-2; \\ x^2+y^2-6, & -2\le xy\le 9. \end{cases}$	Вариант 24 $y=\begin{cases} 3xy-1, & x+y>8; \\ \dfrac{3y^2+x^2}{y^2+1}, & x+y<-2; \\ x^2+4y^2-3, & -2\le x+y\le 8. \end{cases}$
Вариант 25 $y=\begin{cases} \dfrac{x^2+2y^2-3}{3+2x^2}, & x+y<1; \\ 3x^2y+4, & x+y>10; \\ \dfrac{3y+2x}{5+y+x^2}, & 1\le x+y\le 10. \end{cases}$	Вариант 26 $y=\begin{cases} \dfrac{x+xy+3y}{y^2+1}, & x-y>10; \\ 2xy-y^2, & x-y<0; \\ x^2-y^2, & 0\le x-y\le 10. \end{cases}$
Вариант 27 $y=\begin{cases} \dfrac{x+2y-3}{x^2+1+xy}, & xy>12; \\ \dfrac{xy-2y}{x^2+1}, & xy<-4; \\ x^2+3y^2, & -4\le xy\le 12. \end{cases}$	Вариант 28 $y=\begin{cases} \dfrac{2y+3x}{x^2+y^2+5}, & x-y>12; \\ \dfrac{x-y}{x^2+2y^2+1}, & x-y<-4; \\ x^2+3y^2, & -4\le x-y\le 12. \end{cases}$
Вариант 29 $y=\begin{cases} \dfrac{5xy}{x+y^2+1}, & xy<-5; \\ 2x-y, & xy>8; \\ x+3y, & -5\le xy\le 8. \end{cases}$	Вариант 30 $y=\begin{cases} \dfrac{x^2-y^2+5}{3+xy}, & x+y<-7; \\ xy-1, & x+y>10; \\ \dfrac{3y+x}{11y-x^2}, & -7\le x+y\le 10. \end{cases}$

Литература

1 Гук, М. Процессоры Pentium III, Athlon и другие / М. Гук, В. Юров. – СПб. : Питер, 2000. – 379 с.

2 Зубков, С. В. Ассемблер для DOS, Windows и UNIX / С. В. Зубков. – М. : ДМК Пресс, 2000. – 534 с.

3 Программирование на языке ассемблера для персональных ЭВМ : учебное пособие / А. Ф. Каморников [и др.]. – Гомель : ГГУ, 1995. – 95 с.

4 Пустоваров, В. И. Ассемблер : программирование и анализ корректности машинных программ / В. И. Пустоваров. – К. : Издательская группа BHV, 2000. – 480 с.

5 Сван, Т. Освоение Turbo Assembler / Т. Сван. – Киев : Диалектика, 1996. – 540 с.

6 Юров, В. Assembler / В. Юров. – СПб. : Питер, 2001. – 624 с.

7 Юров, В. Assembler : практикум / В. Юров. – СПб. : Питер, 2002. – 400 с.

8 Юров, В. Assembler : специальный справочник / В. Юров. – СПб. : Питер, 2000. – 496 с.

Производственно-практическое издание

Ружицкая Елена Адольфовна

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ASSEMBLER

**Системы счисления,
программная модель микропроцессора,
арифметические команды**

Практическое пособие

Редактор *В. И. Шкредова*
Корректор *В. В. Калугина*

Подписано в печать 05.02.2016. Формат 60x84 1/16.
Бумага офсетная. Ризография. Усл. печ. л. 2,8.
Уч-изд. л. 3,1. Тираж 25 экз. Заказ 58.

Издатель и полиграфическое исполнение:
учреждение образования
«Гомельский государственный университет
имени Франциска Скорины».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/87 от 18.11.2013.
Специальное разрешение (лицензия) № 02330 / 450 от 18.12.2013.
Ул. Советская, 104, 246019, Гомель.