Оглавление

МГТУ им. Н.Э. Баумана

Факультет «Информатика и Системы Управления»

Кафедра ИУ-6 «Компьютерные системы и сети»

ИВАНОВА ГАЛИНА СЕРГЕЕВНА,

НИЧУШКИНА ТАТЬЯНА НИКОЛАЕВНА

Ассемблер IA-32. Вещественная арифметика

Учебное пособие

МОСКВА

2010 год МГТУ им. Баумана

Оглавление

Аннотация

Настоящее учебное пособие содержат сведения, необходимые для написания программ на ассемблере с использованием операций над вещественными числами, и включает:

• описание основных форматов хранения данных в компьютере на базе микропроцессора Intel;

• описание программной модели сопроцессора;

• описания основных команд сопроцессора;

• примеры программ, использующих сопроцессор и методику их составления.

Пособие предназначено для студентов 2 курса кафедры Компьютерные системы и сети, изучающих дисциплину Системное программное обеспечение.

Оглавление

Введение

Важной частью современного процессора фирмы Intel или совместимого с ним является устройство обработки чисел в формате с плавающей точкой. Данное устройство впервые появилось в составе компьютера на базе микропроцессора i8086/88 и получило название *математический сопроцессор*. Выбор названия определялся тем, что устройство было предназначено для расширения возможностей основного процессора и было реализовано в виде отдельной микросхемы. Эта микросхема имела название i8087. Включение устройства в компьютер было не обязательным.

С появлением новых моделей процессоров Intel совершенствовались и сопроцессоры, хотя их программная модель осталась практически неизменной. Для процессоров моделей i80286 и i80386 сопроцессоры также были реализованы в виде отдельных кристаллов и имели обозначения i80287 и i80387 соответственно. А для процессора i80486 уже существовало две реализации: в одной сопроцессор оставался отдельным кристаллом, а в другой – он исполнялся на одном кристалле с основным процессором и становился неотъемлемой частью компьютера. Начиная с процессора типа Pentium, сопроцессор встраивается в основной процессор и преобразуется в *блок операций с плавающей точкой* (Float Point Unit – FPU).

Блок операций с плавающей точкой процессоров семейства IA-32 реализует программно-аппаратную обработку вещественных чисел из диапазона $3.37*10^{-4932}$ ..$1.18*10^{+4932}$ и численные алгоритмы вычисления значений тригонометрических функций, логарифмов и т.д. Это соответствует стандартам Института инженеров по электротехнике и электронике IEEE-754 и IEEE-854, описывающим поддерживаемые форматы данных с плавающей точкой и набор функций, выполняемых устройством.

Оглавление

1 Форматы данных математического сопроцессора

1.1 Представление вещественных чисел в двоичной системе счисления

«*Вещественное* или *действительное* число – математическая абстракция, возникшая из потребности измерения геометрических и физических величин окружающего мира, а также проведения таких операций как извлечение корня, вычисление логарифмов, решение алгебраических уравнений. Если натуральные числа возникли в процессе счета, рациональные – из потребности оперировать частями целого, то вещественные числа предназначены для измерения непрерывных величин» [Википедия].

В математике вещественные числа обычно представляют в виде целой части и десятичной (конечной или бесконечной, периодической или непериодической) дроби, например 1/3 = 0.33(3).

Одной из форм записи вещественных чисел является их представление *в экспоненциальном виде*, в котором отдельно записывают мантиссу числа и его порядок. Однако число при этом может быть записано многими способами, например 7.5 может быть записано как 7.5×10^0 или 0.75×10^1 или 0.075×10^2 и т.д. Поэтому обычно числа в экспоненциальном виде нормализуют. В математике при *нормализованной* записи вещественного числа порядок p выбирается таким, чтобы абсолютная величина мантиссы M была не меньше единицы, но менее десяти $1 \le |M| < 10$ например, 350 записывается как 3.5×10^2. Этот вид записи позволяет легко сравнивать два числа в экспоненциальном виде.

Для представления вещественных чисел в памяти компьютера используют двоичную систему счисления и, соответственно, степени двойки вместо степеней десяти.

Обычную дробь, например, 0.324, в десятичной системе можно представить в виде:

$3/10+2/100+4/1000$ или $3/10^1+2/10^2+4/10^3$, где знаменатели – увеличивающиеся степени числа 10.

В двоичной дроби в качестве знаменателя используются степени 2. Так, двоичную дробь 0.101 можно записать:

Оглавление

$1/2^1+0/2^2+1/2^3$ или 1/2+0/4+1/8 ,

что в десятичной системе равно 0.5+0.0+0.125 = 0.625.

Так же, как и в десятичной системе счисления, в двоичной системе счисления не все дроби можно представить точно. Точно записываются только дроби, знаменатели которых являются степенями 2, например $3/4 = 1/2^1+0/2^2 = 0.11_2$ или $7/16 = 1/2+1/16 = 0.1001_2$.

Кроме того, для представления точного значения, даже если оно существует, может не хватить разрядной сетки формата. Поэтому вещественные числа в памяти компьютера, как правило, *представлены не точно*. Точность их представления определяется количеством бит, отведенных в формате для записи мантиссы. В то время как диапазон представляемых значений определяется количеством бит, отведенных для записи порядка.

При ручном переводе вещественного числа из десятичного представления в двоичное, отдельно переводятся целая и дробная части. Как показывает опыт, перевод целой части вещественного десятичного числа в двоичный формат не представляет труда. А вот преобразование дробной части достаточно сложно.

Существует несколько способов преобразования десятичной дроби в двоичную.

1. Разложение десятичной дроби на сумму дробей вида 1/2+1/4+1/8 +….

Метод заключается в разбиение десятичной дроби на сумму дробей, знаменатель которых кратен степеням двойки.

Например:

$0.625_{10} = 625/1000_{10} = 5/8_{10} = 4/8 + 1/8 = 1/2 + 1/8 = 0.1_2 + 0.001_2 = 0.101_2$

или

$5/8_{10} = 4/8 + 1/8 = 1/2 + 1/8 = 1/2 + 0/4 + 1/8 = 1/2^1 + 0/2^2 + 1/2^3 = 0.101_2$.

В таблице 1 приведены несколько примеров такого преобразования.

Таблица 1 – Примеры преобразования дробей

Десятичная дробь	Разложение	Двоичная дробь
1/2	1/2	0.1
1/4	1/4	0.01
3/4	1/2+1/ 4	0.11
1/8	1/8	0.001
7/8	1/2+1/4+1/8	0.111
3/8	1/4+1/8	0.011
1/16	1/16	0.0001
3/16	1/8 +1/16	0.0011

Оглавление

5/16	1/ 4 + 1/16	0.0101

Однако большинство десятичных дробей нельзя представить в виде суммы дробей, чьи знаменатели раскладываются по степеням двойки. Так 1/5 в виде суммы дробей со знаменателями – степенями двойки представить нельзя. В этом случае строится приближенное представление, для получения которого следует использовать другие приемы перевода.

2. Преобразование методом деления числителя дроби на ее знаменатель.

Вещественное десятичное число можно представить в виде двоичного числа, если сначала преобразовать числитель и знаменатель в двоичную систему, а затем поделить их «в столбик» по правилам двоичного деления.

Пример. Десятичное число 0.5 можно представить так:

$0.5=5/10=0101_2/1010_2$.

Далее выполняется деление двоичных чисел:

```
 0101   |1010
-01010    0.1
  1010
     0
```

Как видно из примера, если делимое меньше делителя, в частное записывается 0, а делимое сдвигается влево на один разряд, с добавлением в младший разряд нуля. Если полученное число меньше делителя, в частное опять пишется 0 и опять производится сдвиг влево.

Так делают до тех пор, пока делимое не станет больше делителя. Если делимое больше делителя – из него вычитается делимое, а в частное пишется 1. Остаток от деления рассматривается, как новый делитель.

Если остаток 0, то процесс перевода завершается. В противном случае остаток опять сдвигается влево на один разряд, и процедура повторяется до тех пор, пока в остатке не окажется 0 или количество двоичных разрядов частного не превысит количество разрядов, предусмотренных форматом для представления числа.

Оглавление

Пример. Перевод числа 0.2 в двоичное число: $2/10=10_2/1010_2$.

```
 0010 | 1010
        0.00110011(0011)
_0010000
 __1010
  _01100
   _1010
    _0010000
     ___1010
        _01100
         _1010
          _0010000
            __1010
              _01100
               _1010
                _0010000   …….
```

Как видно из примера, выделенный фрагмент начинает повторяться бесконечное количество раз, т.е. мы получили бесконечную периодическую двоичную дробь.

3. Перевод умножением десятичной дроби на 2.

Этот метод является стандартным методом перевода десятичной дроби в двоичное представление. Алгоритм перевода следующий:

а) последовательно выполняют умножение исходной десятичной дроби и получаемых дробных частей произведений на 2 до тех пор, пока не будет получена нулевая дробная часть или достигнута требуемая точность вычислений;

б) целые части произведения записывают в том порядке, в котором они получаются.

Например:

а) перевод дроби 0.75:

$0.75 \times 2 = 1.5 \rightarrow 1$

$0.5 \times 2 = 1 \rightarrow 1$

$0 \times 2 = 0$

Результат перевода: $0.75_{10}=0.11_2$

б) перевод десятичной дроби 0.2:

$0.2 \times 2 = 0.4 \rightarrow 0$

$0.4 \times 2 = 0.8 \rightarrow 0$

$0.8 \times 2 = 1.6 \rightarrow 1$

Оглавление

$0.6 \times 2 = 1.2 \rightarrow 1$

$0.2 \times 2 = 0.4 \rightarrow 0$

$0.4 \times 2 = 0.8 \rightarrow 0$

$0.8 \times 2 = 1.6 \rightarrow 1$

.....

Результат перевода: 0.2_{10}=0.0011001…….

Оглавление

1.2 Формат IEEE представления двоичных чисел с плавающей точкой

В основе всех форматов, закрепленных стандартами IEEE, лежит представление числа в виде мантиссы со знаком и порядка, которое определяется формулой:

$A=(\pm M)*N^{(\pm p)}$,

где M – мантисса числа;

N – основание системы счисления, представленное целым положительным числом;

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

Архитектура процессора, как правило, накладывает ограничения на представление вещественных чисел. Для процессоров Intel эти ограничения сформулированы следующим образом:

- основание системы счисления N = 2;

- чтобы с максимальной точностью хранить в памяти двоичное число с плавающей точкой, его мантисса M должна быть представлена в нормализованном виде;

- чтобы не хранить знак порядка, число, характеризующее порядок числа, должно быть положительным.

Двоичное нормализованное представление числа определяется выполнением условия

$1_{10} \le |M_{10}| < 2_{10}$ или в двоичной системе счисления $1.0..00_2 \le |M_2| < 10.0..00_2$,

то есть единственная цифра целой части мантиссы должна быть значащей (единицей), а порядок p, соответственно, таким, чтобы это условие выполнялось. Таким образом *число считается нормализованным, если слева от десятичной точки находится только один двоичный разряд, значение которого равно 1.*

Процесс нормализации вещественного двоичного числа ничем не отличается от нормализации десятичного вещественного числа. Например, десятичное число 1234.567 в нормализованном виде выглядит так: $1.234567*10^3$. Как видно, в процессе нормализации десятичная точка переносится влево или вправо так, чтобы перед ней находилась только одна десятичная цифра. При этом значение показателя степени определяет количество цифр, на которое нужно передвинуть десятичную точку вправо (если показатель положителен) или влево (если показатель отрицателен), чтобы получить истинное значение числа.

Оглавление

Аналогично число 1101.101 после нормализации будет выглядеть как: $1.101101*2^3$. То есть, при нормализации точка была перенесена на 3 разряда влево, а значит, для получения исходного значения, полученный результат следует умножить на 2^3.

Примеры нормализации двоичных чисел:

а) 11011.101 = 1.1011101×2^4

б) 0.00001011 = 1.011×2^{-5}

в) 1.0101 = 1.0101×2^0

г) 100000010.0 = 1.000000100×2^8

Для выполнения последнего ограничения процессора, касающегося строго положительного «порядка» числа, в памяти хранится не сам порядок, а его *характеристика*, которая получается добавлением к порядку некоторого фиксированного смещения. Фиксированное смещение для каждого из трех форматов вещественных чисел имеет свое значение, которое определяется количеством разрядов, отведенных в формате на порядок. В процессе вычислений по характеристике *аппаратно* определяется порядок.

Таким образом значение A вещественного числа определяется формулой:

$$A = (-1)^s\times2^{q-f}\times M,$$

где s – значение знакового разряда: 0 – число больше нуля, 1 – число меньше нуля;

q = p+f – характеристика порядка p, которая представляет из себя сумму порядка и некоторой константы f;

M – нормализованная мантисса числа.

В соответствии со стандартом данные вещественного типа могут представляться в одном из трех форматов: короткое – с одинарной точностью, длинное – с двойной точностью и расширенное (см. таблицу 2).

Оглавление

Таблица 2 – Форматы представлений вещественных чисел

Формат	Описание
Короткое вещественное число	**32 бита:** первый бит – знак мантиссы, 8 бит – для представления характеристики и 23 бита – для представления мантиссы. С помощью этого формата можно представить нормированные числа в диапазоне приблизительно от 2^{-126} до 2^{+127}.
Длинное вещественное число	**64 бита:** первый бит – знак мантиссы, 11 бит – характеристика и 52 бита – мантисса. С помощью этого формата можно представить нормированные числа в диапазоне приблизительно от 2^{-1022} до 2^{+1023}.
Расширенное вещественное число	**80 бит:** первый бит – знак мантиссы, 16 бит – характеристика и 63 бита – мантисса. С помощью этого формата можно представить нормированные числа в диапазоне приблизительно от 2^{-16382} до 2^{+16383}.

Если значение знакового бита равно 1, то число считается отрицательным, если 0 – то положительным. Число нуль считается положительным.

Нормализованное двоичное число согласно правилу нормализации всегда имеет целую часть, равную 1. Поэтому при его представлении в памяти появляется возможность считать первый разряд единичным по умолчанию и учитывать его наличие только на аппаратном уровне. Это дает возможность увеличить диапазон представляемых чисел, так как появляется лишний разряд, который можно использовать для представления мантиссы числа. Но это справедливо только для короткого и длинного форматов вещественного числа. Расширенный формат, как внутренний формат представления числа любого типа в сопроцессоре, содержит целую единицу в явном виде.

В таблице 3 представлены основные характеристики каждого из указанных форматов представления вещественных чисел.

Таблица 3 – Характеристики форматов вещественных чисел

	Короткий	Длинный	Расширенный
Длина числа, бит	32	64	80
Размерность мантиссы, бит	24	53	64
Диапазон представляемых значений	10^{-38} .. 10^{+38}	10^{-308} .. 10^{+308}	10^{-4932} .. 10^{+4932}
Размерность характеристики q, бит	8	11	16
Фиксированное смещение f	+127	+1023	+16383
Диапазон порядка p	-126 .. 127	-1022 .. +1023	-16382 .. +16383
Диапазон характеристики q	0 .. 255	0 .. 2047	0 .. 32767

Оглавление

Как следует из таблицы, все положительные порядки имеют в двоичном представлении характеристики старший бит, равный единице, а отрицательные нет. Например:

а) p = -1, q= -1+127=126 → характеристика=01111110;

б) p=1, q=1+127=128 → характеристика=10000000.

Таким образом, в *старшем бите характеристики скрыт знак порядка вещественного числа.*

Обобщенный алгоритм преобразования вещественного десятичного числа в двоичное число с плавающей точкой формата IEEE. Этот алгоритм описывает последовательность действий, которые нужно предпринять, для преобразования вещественного числа в двоичное число формата IEEE.

1. Представить целую часть вещественного числа в двоичном виде и поставить после нее десятичную точку.

2. Преобразовать дробную часть вещественного числа в двоичный формат, используя один из приведенных выше методов для определения значений всех битов мантиссы, предусмотренных форматом.

3. Записать полученное значение дробной части после десятичной точки. Если значение мантиссы меньше выделенного под нее количества разрядов, то дополнить ее незначащими нулями справа до предусмотренного форматом размера.

4. Нормализовать полученное двоичное число, определив значение показателя степени.

5. К показателю степени прибавить фиксированное число в соответствии с форматом, и представить его в двоичном виде. В результате будет получено значение характеристики числа, используемой в выбранном формате.

6. Записать значение характеристики в соответствующие биты формата перед нормализованной мантиссой, отбросив единицу целой части мантиссы при использовании короткого и длинного форматов.

7. Если число положительное, то в самый старший разряд представления следует записать 0, если отрицательное – то 1.

Пример. Преобразовать вещественное число 45.56 в двоичное число с плавающей точкой короткого формата.

В соответствии с алгоритмом, сначала получим двоичное представление целой части. Для этого, используя стандартный алгоритм, число 45 будем последовательно делить на 2.

Оглавление

```
_45|2
 44 _22|2
  1  22  _11|2
      0  10  _5|2
          1   4  _2|2
              1   2  _1|2
                  0   0  0
                         1
```

Запишем остатки от деления в обратном порядке 101101 – это и есть двоичное представление числа 45. Нетрудно проверить правильность перевода:

$101101_2=1*2^5+0*2^4+1*2^3+1*2^2+0*2^1+1*2^0=32+8+4+1=45_{10}$.

Теперь переведем в двоичный формат дробь 0.56, используя метод умножения дробной части на 2.

0.56*2= **1**.12
0.12*2= **0**.24
0.24*2= **0**.48
0.48*2= **0**.96
0.96*2= **1**.92
0.92*2= **1**.84
0.84*2= **1**.68
0.68*2= **1**.36
0.36*2= **0**.72
0.72*2= **1**.44
0.44*2= **0**.88 …….

В соответствии с методом выделенные разряды целой части дроби являются разрядами дроби. В результате получим дробную часть 100011110101110001.

Запишем полученную дробь с десятичной точкой

101101**.**100011110101110001

Нормализуем:

$1.10110110001111010111000 01*2^5$ → показатель p=5.

Добавляем к показателю константу f=127 (для короткого формата) и получаем характеристику q=132.

Переводим число 132_{10} в двоичный вид:

Оглавление

```
_132|2
 132 _66 |2
   0  66  _33 |2
       0   32  _16|2
           1   16   _8|2
               0    8    _4 |2
                    0    4   _2 |2
                         0    2   _1  |2
                              0   0    0
                                  1
```

Получаем: 10000100_2.

Число 45.56 – положительное, поэтому знаковый разряд установим в 0.

Теперь все объединяем. Только следует отбросить целую часть мантиссы, так как для короткого и длинного формата она учитывается аппаратно, как было сказано ранее.

В результате, число 45.56 в формате короткого вещественного числа будет выглядеть так:

0100 0010 0011 0110 0011 1101 0111 0001
+_____/ ____/ _____/
Знак q=132 45 . (усл. точка) 56

Так как при отладке все двоичные числа визуализируются в шестнадцатеричном формате, для уменьшения объема выводимой информации и ее более легкого восприятия, переведем полученное число в шестнадцатеричное представление. Для этого разобьем двоичное число на тетрады и каждую переведем в 16-тиричную цифру. После перевода число будет выглядеть так:

42 36 3d 71.

Учитывая, что в архитектуре Intel принят обратный порядок следования байт при записи числа в памяти в соответствии с принципом «младший байт по младшему адресу», в памяти мы увидим число:

71 3d 36 42.

В программе вещественные числа описываются с помощью директив:

DD – для вещественного числа короткого формата;

DQ – для вещественного числа длинного формата;

DT – для вещественного числа расширенного формата.

Оглавление

При этом вещественная константа может быть представлена как в форме с фиксированной точкой, так и в форме числа с плавающей точкой. При компиляции это число будет переведено в соответствующий формат IEEE согласно приведенному выше алгоритму.

Например, для записи числа 45.56 в коротком формате можно задать директиву

`DD  45.56` или **`DD  45.56e0`** или **`DD  0.4556e2`**.

При просмотре этой константы в памяти мы увидим число **71 3d 36 42.**

Для задания этого же числа в длинном формате следует указать:

`DQ  45.56` или **`DQ  45.56e0`**

Истинное значение числа после перевода в формат будет таким:

40 46 c7ae 14 7a e1 47

В памяти соответственно мы увидим:

47 e1 7a 14 ae c7 46 40

Что это так, можно убедиться, выполнив преобразование самостоятельно, в соответствии с алгоритмом.

Определение числа 45.56 в расширенном формате числа выполняется директивой:

`DT  45.56`

Истинное значение числа будет выглядеть как:

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

В памяти число будет в виде:

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

А вот формирование двоичного эквивалента интересно посмотреть.

Число будет иметь следующие значения битов (в соответствии с алгоритмом):

Старший бит – 0 (знак +).

Целая часть $45_{10} = 101101_2$.

Дробная часть

0.56= 10 0011 1101 0111 0000 1010 0011 1101 0111 0000 1010 1110 0001 0111 0001.

45.56_{10}=101101. 10 0011 1101 0111 0000 1010 0011 1101 0111 0000

1010 1110 0001 0111 0001_2.

Оглавление

Нормализуем:

101101. 10 0011 1101 0111 0000 1010 0011 1101 0111 0000 1010 1110 0001 0111 0001=

1.01101 10 0011 1101 0111 0000 1010 0011 1101 0111 0000 1010 1110 0001 0111 0001*2^5.

Характеристика q =16388 $_{10}$ = 16383+5 = 100000000000100_2;

В результате, после объединения получаем:

0100000000000100**1**0110110001111010111000010100011110101110000101011100001011100 01
+_____q=16388_/ _45_/ . _____56_____/

Жирным шрифтом выделен старший разряд мантиссы, который явно присутствует в этом формате. Как было отмечено, это связано с тем, что именно расширенный формат используется для внутреннего представления всех остальных форматов данных. Кроме того, он является единственным форматом представления чисел в регистрах сопроцессора. Само преобразование выполняется автоматически при загрузке числа в стек сопроцессора.

Оглавление

1.3 Другие форматы данных, обрабатываемые сопроцессором

Кроме трех основных вещественных форматов, для обработки которых собственно и разрабатывался сопроцессор, последний может работать и с другими форматами, хотя и менее эффективно. Сопроцессор может обрабатывать:

А) Целые числа:

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

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

Б) Специальные численные значения:

- денормализованные вещественные числа;

- нуль;

- положительные и отрицательные значения бесконечности;

- нечисла;

- неопределенности и неподдерживаемые форматы.

Двоичные целые числа. Сопроцессор работает с тремя типами целых чисел (см. таблицу 4). Однако при выборе целого формата представления данных следует помнить, что работа сопроцессора с этим форматом не эффективна из-за необходимости выполнения дополнительного преобразования целых чисел в их внутреннее представление в виде эквивалентного вещественного числа в расширенном формате.

Таблица 4 – Характеристики форматов целых чисел

Формат	Размер	Диапазон значений
Целое слово	16	-32768 .. +32767
Короткое целое	32	$-2*10^9$ .. $+2*10^9$
Длинное целое	64	$-9*10^{18}$ .. $+9*10^{18}$

В программе целые числа описываются директивами **DW**, **DD**, **DQ** соответственно. Например:

```
Cislo_w    DW   5;   представление в памяти  05 00
Cislo_dd   DD   5;   представление в памяти  05 00 00 00
Cislo_dq   DQ   5;   представление в памяти  05 00 00 00 00 00 00 00
```

Оглавление

Упакованные десятичные числа. Сопроцессор может обрабатывать только один формат упакованных десятичных чисел. Во внутреннем представлении такое число занимает 80 бит или 10 байт. Старший 10-й байт в старшем бите содержит знак числа. Остальная часть этого байта игнорируется. Остальные 9 байт содержат по две десятичных цифры. Соответственно в регистры сопроцессора можно поместить только 18 десятичных цифр.

Упакованные десятичные числа также представляются в сопроцессоре в расширенном формате. В программе упакованные десятичные числа описываются директивой DT.

Например, число 5678937 в упакованном формате описывается так:

`Chislo_dt DT  5678937;` представление в памяти 07 93 78 56 00 00 00 00 00 00

Специальные значения. Несмотря на большой диапазон вещественных чисел, представимых в формате сопроцессора, бесконечное количество чисел находится за рамками представимого диапазона. Для получения возможности реагировать на ситуации, в которых могут быть получены значения из непредставимого диапазона, в сопроцессоре предусмотрены специальные комбинации бит, называемые *специальными численными значениями*.

К специальным значениям относятся:

- денормализованные вещественные числа,
- отрицательная и положительная бесконечности,
- нечисла;
- неопределенности,
- значения в неподдерживаемых форматах,
- нуль.

Для представления специальных значений зарезервированы минимальная 000..00 и максимальная 111..11 характеристики.

Нуль. Значение нуля относится к специальным. Это делается из-за того, что нуль выделяется среди корректных вещественных значений, формируемых как результат некоторой операции. Кроме того, нуль может формироваться как реакция сопроцессора на определенную ситуацию.

Нуль представляется с нулевой характеристикой и нулевой мантиссой:

0 0000000 00000000 000000000 000000000.

Денормализованные вещественные числа. Денормализованные числа – числа, значение которых меньше минимально представимого в нормализованном виде в формате. По мере

приближения к нулю значение мантиссы уменьшается и наступает момент, когда разрядной сетки, отведенной под характеристику, становится мало. В этот момент значение характеристики обращается в нуль. Число, при достижении которого это происходит на самом деле отлично от нуля, так как между истинным нулем и минимальным нормализованным находится еще бесконечное множество чисел. Именно эти числа и представляют собой денормализованные числа. Однако диапазон денормализованных чисел не безграничен и определяется количеством разрядов, отведенных под мантиссу.

Денормализованное число имеет нулевую характеристику и ненулевую мантиссу, например:

0 0000000 00000100 00000111 00000000

Бесконечности. Сопроцессор имеет средства для представления бесконечности. Среди причин, приводящих к формированию значения бесконечности, в первую очередь следует назвать переполнение и деление на нуль.

Бесконечность кодируются с максимальным значением характеристики 111..11 и мантиссой равной 1. Соответственно различают положительную и отрицательную бесконечности:

$+\infty$ **0 1111111 10000000 00000000 00000000**

$-\infty$ **1 1111111 10000000 00000000 00000000**

Нечисла. К нечислам относятся такие битовые последовательности, которые не совпадают ни с одним из рассмотренных выше форматов.

Нечисла представляются с любым знаком, максимальной характеристикой и любой мантиссой кроме 1, например:

0 1111111 10000100 00000000 00000000

Неопределенность. Неопределенностью называют результат недействительной операции:

0 1111111 10000000 00000000 00000000

Неподдерживаемые форматы. Существует достаточно много битовых наборов, которые можно представить в расширенном формате. Для большинства из этих форматов формируется исключение «недействительная операция».

Оглавление

Контрольные вопросы

1. Объясните, почему в двоичной системе счисления не все вещественные числа могут быть представлены точно.

Ответ.

2. Перечислите методы перевода вещественного числа в двоичный формат.

Ответ.

3. Какие форматы вещественных чисел предусматривает IEEE?

Ответ.

4. Какие еще форматы чисел обрабатывает вещественный сопроцессор?

Ответ.

Оглавление

2 Программная модель сопроцессора

Программная модель сопроцессора с точки зрения программиста представляет собой совокупность регистров, каждый из которых имеет свое функциональное назначение. В программной модели сопроцессора можно выделить три группы регистров: регистры стека, служебные регистры и регистры данных (см. рисунок 1).

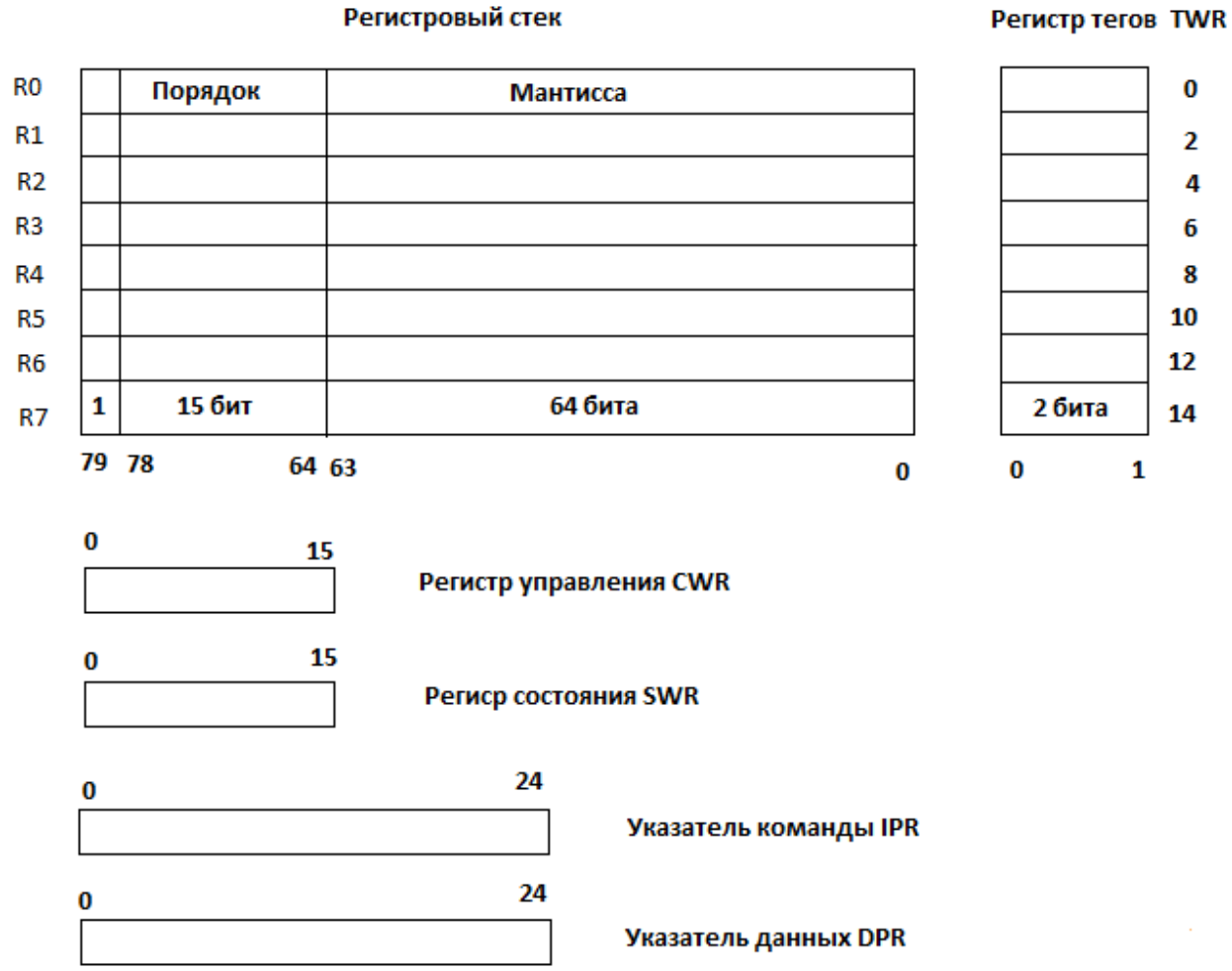

Рисунок 1 – Регистры сопроцессора

Такая организация характерна для устройств, специализирующихся на обработке вычислительных алгоритмов. Реализация численных алгоритмов на основе регистрового стека позволяет получить существенный выигрыш в скорости вычислений.

Оглавление

2.1 Регистры стека

Команды сопроцессора не оперируют физическими номерами регистров стека R0 – R7. Вместо этого они используют логические номера этих регистров St(0) – St(1). Именно с помощью логических номеров реализуется относительная адресация регистрового стека сопроцессора.

По мере записи в стек указатель его вершины движется по направлению к младшим номерам физических регистров. Так, если текущей вершиной стека был регистр R0, то после записи очередного значения в стек сопроцессора текущей вершиной стека станет физический регистр R7. Логические же номера «плавают» вместе с текущей вершиной стека. Логический регистр St(0) всегда ассоциирован с вершиной стека. Таким образом реализуется принцип кольца.

Стековая организация регистров сопроцессора позволяет не привязываться к аппаратным ресурсам – физическим номерам регистров сопроцессора, а оперировать логической нумерацией, поддерживаемой на уровне системы команд. При этом не имеет значения номер физического регистра, в который помещаются данные, так как определяющим является порядок следования данных в стеке.

Оглавление

2.2 Служебные регистры

К служебным относятся три регистра – регистр состояния сопроцессора, управляющий регистр сопроцессора и регистр слова тегов.

Регистр состояния сопроцессора swr (Status Word Register) отражает текущее состояние сопроцессора после выполнения команды. В регистре содержатся поля, позволяющие определить: какой регистр является вершиной стека, какие исключения возникли после выполнения команды, какие особенности выполнения команды. Регистр состояния включает следующие биты:

а) *6 флагов исключительных ситуаций* – все возможные исключения сведены к шести типам, каждому из которых соответствует один бит регистра состояния (флаг).

Биты 0-5 устанавливаются при наступлении особых случаев, если они не маскированы:

- бит 0 – недействительная операция (IE);
- бит 1 – ненормализованный операнд (DE);
- бит 2 – деление на ноль (ZE);
- бит 3 – переполнение (OE);
- бит 4 – антипереполнение (UE);
- бит 5 – потеря точности (PE);

б) *бит sf (Stack Fault)* – ошибка работы стека сопроцессора, бит устанавливается в единицу, если возникла одна из трех исключительных ситуаций. В частности его установка информирует о попытки записи в заполненный стек или чтения из пустого стека.

После анализа этого бита его следует установить в 0.

SF=1, C1=1 – переполнение стека;

SF=1, C1=0 – чтение из пустого стека;

в) *бит es (Error Summary)* – суммарная ошибка сопроцессора – устанавливается в единицу, если возникает любая из шести перечисленных выше исключительных ситуаций;

г) *4 бита C0 –C3 – коды условия (Condition Code)* – назначение этих битов аналогично флагам в регистре флагов основного процессора – отразить результат выполнения операции. Интерпретация битов:

бит 8 – C0 (**CF**); бит 9 – C1; бит 10 – C2 (**PF**); бит 14 – C3 (**ZF**);

Оглавление

д) *биты 11-13 – трехбитовое поле top* – содержит указатель регистра текущей вершины стека;

е) *бит 15 – бит занятости сопроцессора (B)*.

Регистр управления cwr (Control Word Register) определяет особенности обработки численных данных. Он состоит из:

а) *шести масок исключений* – маски предназначены для маскирования исключительных ситуаций, возникновение которых фиксируется с помощью шести флагов регистра состояния. Если какая то из шести масок установлена в 1, то соответствующее исключение будет обрабатываться самим сопроцессором. Если маски установлены в 0, то при возникновении соответствующего исключения будет возбуждено прерывание. При этом операционная система должна содержать соответствующий обработчик.

Биты 0..5 – маска особых случаев – используется для запрета прерывания в случае, если:

- бит 0 – обнаружена недействительная операция (**IM=0**);
- бит 1 – обнаружен ненормализованный операнд (**DM=1**);
- бит 2 – зафиксировано деление ненулевого числа на ноль (**ZM=1**);
- бит 3 – обнаружено переполнение (**OM=1**);
- бит 4 – обнаружено антипереполнение (исчезновение порядка **UM=1**);
- бит 5 – обнаружена потеря точности (**PM=1**);

б) *поле управления точностью* (**PC**) – предназначено для выбора длины мантиссы.

Биты 8-9 –

00 – 24 бита; 10 – 53 бита; 11 – 64 бита.

По умолчанию значение поля устанавливается pc=11;

в) *поля управления округлением* (**RC**) – позволяет управлять процессом округления чисел во время работы сопроцессора. Необходимость округления может возникнуть, если в процессе выполнения команды получается непредставимый результат, например, периодическая дробь. Установка поля округления позволит выполнить округление в нужную сторону.

Биты 10-11 задают режим округления:

00 – к ближайшему целому;

01 – округление в меньшую сторону;

10 – округление в большую сторону;

Оглавление

11 – отбрасывание дробной части (используется для приведения числа к форме, которая используется в операциях целочисленной арифметики).

г) *бит 12* **–** интерпретации бесконечности (**IC=1**):

0 – беззнаковая бесконечность; 1 – бесконечность со знаком.

Регистр тегов twr (Tag Word Register) **–** используется для контроля состояния каждого из регистров R0 – R7. Команда сопроцессора использует этот регистр для определения, например, возможности записи в эти регистры.

Регистр отводит 2 бита на каждый регистр стека:

00 – в регистре не нулевое действительное число;

01 – в регистре истинный ноль;

10 – в регистре не число или бесконечность;

11 – регистр пуст.

Оглавление

2.3 Регистры указателей

Программная модель сопроцессора оперирует двумя регистрами указателей: указатель команды и указатель данных.

Регистр указатель команды ipr (Instraction Point Register) содержит адрес команды, вызвавшей особый случай, и 11 бит команды.

Регистр указатель данных dpr (data Point Register) содержит адрес операнда (если использовался) команды, вызвавшей особый случай.

Как было отмечено ранее, все регистры программной модели сопроцессора программно доступны. Однако, к одним из них доступ возможен с помощью специальных команд, предусмотренных в системе команд сопроцессора. К другим же регистрам доступ возможен только при выполнении дополнительных действий, так как специальных команд для такого доступа не предусмотрено.

Оглавление

Контрольные вопросы

1. Назовите регистры стека. Объясните, как они реализованы и почему?

Ответ.

2. Как используются служебные регистры?

Ответ.

3. Для чего в сопроцессоре применяются регистры указателей?

Ответ.

Оглавление

3 Система команд сопроцессора

Система команд сопроцессора включает около 80 машинных команд. Она отличается большой гибкостью в выборе вариантов задания команд, реализующих определенную операцию, и их операндов. Минимальная длина команды – 2 байта.

Все команды условно можно разбить на пять групп:

- передачи данных;
- арифметические;
- сравнения;
- трансцендентных операций;
- управления.

Мнемоническое обозначение команд сопроцессора характеризует особенности их работы. В связи с этим в мнемонике приняты следующие соглашения:

- первая буква всегда **F**;
- вторая буква определяет тип операнда в памяти, с которым работает команда:

I – обозначает операцию с целым числом из памяти;

B – операцию с десятичным числом из памяти;

при отсутствии этих букв – операция выполняется с вещественными числами;

- предпоследняя или последняя букв **R (reversed)** указывает обратный порядок операндов (для вычитания и деления, так как для этих команд очень важен порядок следования операндов);
- последняя буква **P** идентифицирует команду, последним действием которой является извлечение из стека операнда.

Прежде чем перейти к описанию форматов команд, следует вспомнить, что все они работают с данными, которые располагаются в памяти. Для этого под данные необходимо зарезервировать память в соответствии с форматом данных. Предусмотрены следующие директивы резервирования памяти для команд сопроцессора.

- 16 бит: DW, WORD, SWORD;
- 32 бита: DD, DWORD, SDWORD, REAL4 (вещественное);
- 64 бита: DQ, QWORD, REAL8;

Оглавление

– 80 бит: DT, TBYTE, REAL10.

При выполнении команды выборки из памяти сопроцессор проверяет тип зарезервированной памяти. Значением типа является длина в байтах, поэтому *резервирование памяти должно совпадать с ее использованием*.

Если обращение к памяти выполняется без указания имени поля, т.е. с адресацией через регистры, то должны использоваться специальные описатели, например:

```
word ptr;   dword ptr;    qword ptr;    tword ptr
```

Например:

```
FLD   QWORD PTR [EBX]
```

Оглавление

3.1 Команды передачи данных

Эта группа команд предназначена для организации обмена между регистрами стека, вершиной стека и ячейками памяти. С помощью этих команд осуществляются все перемещения значений операндов в сопроцессор и из него. Для каждого их трех типов данных, с которыми может работать сопроцессор, определена своя группа команд. Главная функция всех команд загрузки данных в сопроцессор является их преобразование в единое представление в виде расширенного вещественного числа. Все операции сохранения данных в память выполняют обратное преобразование.

1. **Команды загрузки чисел в ST(0):**
   - **FLD <источник>** ; вещественных чисел из памяти или стека
   - **FILD <источник>** ; целых чисел из памяти
   - **FBLD <источник>** ; десятичных чисел из памяти

Примеры:

```
FLD     ST(0)     ; копирует вершину стека
FLD     QWORD PTR [EBX] ; загружает длинное вещественное
FILD    WORD PTR [EDI]  ; загружает целое число
```

2. **Команды сохранения без извлечения из стека**
   - **FST <приемник>** ; копирует вещественное число из ST(0) в память
   - **FIST <приемник>** ; копирует целое число из ST(0) в память

Примеры:

```
FST     ST(5)     ; копирует число из ST(0) в ST(5)
FST     QWORD PTR [EBX] ; копирует вещ. число в память
FIST    WORD PTR [EDI] ;  копирует целое число в память
```

3. **Команды записи в память с извлечением из стека**
   - **FSTP <приемник>;** перемещает значение ST(0) в память (вещественное)
   - **FISTP <приемник>;** перемещает значение ST(0) в память (целое)
   - **FBSTP <приемник>;** перемещает значение ST(0) в память (десятичное)

Примеры:

Оглавление

```
FSTP  ST(5)      ; перемещение в регистр стека (вещественное)
FISTP QWORD PTR [EBX] ; перемещение из стека в память (целое)
FBSTP P1         ; перемещение из стека в память (десятичное)
```

4. Команда обмена

- **FXCH [<приемник>]** ***;*** меняет местами ST(0) и приемник, если приемник не указан, то меняются местами ST(0) и ST(1)

Пример:

```
FXCH   ST(5)     ; меняет местами ST(0) и ST(5)
FSQRT            ; извлечение квадратного корня из ST(0)
FXCH   ST(5)     ; меняет местами ST(0) и ST(5)
```

5. Команды загрузки констант

- **FLDZ** ***;*** загрузка нуля
- **FLD1** ***;*** загрузка единицы
- **FLDPI** ; загрузка π
- **FLDL2T** ***;*** загрузка двоичного логарифма десяти
- **FLDLG2** ; загрузка десятичного логарифма двух
- **FLDLN2** ; загрузка натурального логарифма двух

Оглавление

3.2 Арифметические команды

Данная группа команд реализует четыре основные арифметические операции – сложение, вычитание, умножение и деление. С точки зрения типов операндов, арифметические команды можно разделить на работающие с целыми и вещественными операндами.

Форматы основных арифметических команд:

Fop [ST(1),ST] ***;*** стековая (всегда с извлечением P)

Fop ST(i),ST или Fop ST,ST(i) ; регистровая

FopP ST(i),ST ***;*** регистровая с извлечением

Fop [ST,]<операнд2> ***;*** вещественный операнд в памяти

Fop [ST,]<операнд2> ***;*** целочисленный операнд в памяти

где op = ADD **<операнд1>=<операнд1>+<операнд2>**

SUB **<операнд1>=<операнд1>-<операнд2>**

SUBR **<операнд1>=<операнд2>-<операнд1>**

MUL **<операнд1>=<операнд1>*<операнд2>**

DIV **<операнд1>=<операнд1>/<операнд2>**

DIVR **<операнд1>=<операнд2>/<операнд1>**

Примеры: **FADD** ***; ST←ST+ST(1)*** – в стеке только результат

FADDP ST(5),ST ***; ST(5)←ST(5)+ST*** и извлечение

Дополнительные арифметические команды

- **FSQRT** ***;*** извлечение квадратного корня
- **FSCALE** ***;*** масштабирование ST(0)←ST(0)*2ST(1), ST(1) – целое число
- **FPREM** ***;*** вычисляет частичный остаток ST(0)← ST(0) - q*ST(1),
- где q – целая часть результата ST(0)/ST(1)
- **FPREM1** ***;*** вычисляет частичный остаток ST(0)← ST(0) - q*ST(1),
- где q – ближайшее к ST(0)/ST(1) целое число
- **FPNDINT** ***;*** округление до целого
- **FXTRACT** ***;*** расцепляет число на

- порядок, который заменяет число в ST(1),

Оглавление

- мантиссу, которая помещается в ST(0)

- **FABS** ***;*** получение модуля числа
- **FCHS** ***;*** изменение знака числа

Оглавление

3.3 Команды сравнения

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

- **FCOM [<операнд>]** ; сравнение чисел
- **FCOMP [<операнд>]** ***;*** сравнение чисел и одно извлечение
- **FCOMPP [<операнд>]** ***;*** сравнение чисел и два извлечения из стека
- **FICOM <операнд>** ; сравнение с целым числом
- **FICOMP <операнд>** ***;*** сравнение с целым и извлечение из стека
- **FUCOM <регистр>** ***;*** сравнение с регистром
- **FUCOMP <регистр>** ***;*** сравнение с регистром и извлечение
- **FUCOMPP <регистр>** ***;*** сравнение с регистром и два извлечения
- **FTST** ***;*** сравнение с нулем и замена источника на нуль
- **FXAM** ***;*** анализ операнда

В результате работы команд сравнения в регистре состояния устанавливаются значения битов кода условия (см. таблицу 4).

Таблица 4 – Установка кодов условий при сравнении операндов

Условие	C3	C2	C1	C0
ST(0) > <операнд2> или ST(1)	**0**	0	X	**0**
ST(0) < <операнд2> или ST(1)	**0**	0	X	**1**
ST(0) = <операнд2> или ST(1)	**1**	0	X	**0**
Не сравнимы	**1**	**1**	X	**1**

Чтобы передать результаты сравнения из сопроцессора основному процессору для обработки полученных кодов командами условного перехода основного процессора, нужно записать биты условия в регистр процессора ***eflags***.

Для этого:

1. Сразу после сравнения код необходимо сохранить в регистре AX или в памяти. Для этого в системе команд сопроцессора есть специальная команда

```
FSTSW AX   ; сохранить слово состояния в регистре AX
```

Оглавление

2. Затем значения нужных бит извлекаются и анализируются основным процессором. Это можно сделать двумя способами:

а) загрузить коды во флажковый регистр:

SAHF ; команда копирует биты: **C3** в **ZF**, **C2** в **PF**, а **C0** в **CF**

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. Далее для перехода можно использовать команды JE, JNE, JA, JB, JBE, а для проверки операндов нечисел – команду JP;

б) можно использовать команду **TEST AX,<константа кода>**

Константы, используемые для анализа комбинации кодов, приведены в таблице 5.

Таблица 5 – Константы для анализа комбинации кодов

Результат сравнения	Константа	Переход	B=TOP?0< XXXXXXXX
ST > операнд	4500h	JZ	01000101 00000000
ST < операнд	0100h	JNZ	00000001 00000000
ST = операнд	4000h	JNZ	01000000 00000000
Не сравнимы	0400h	JNE	00000100 00000000

Пример. Проверка условия |p| < eps выхода из итерационного цикла

```
cycl:  ...
       fld   p
       fabs
       fcom  eps
       fstsw AX
       and   AX,0700h ;  выделяем биты C0, C1, C2
       cmp   AH,1h
       je    output
       jmp   cycl
```

Оглавление

3.4 Команды трансцендентных функций

Сопроцессор имеет ряд команд, предназначенных для вычисления значений тригонометрических, а также логических и показательных функций. Это значительно облегчает работу программиста. Однако следует помнить, что значение всех операндов команд, вычисляющих тригонометрические функции, задаются в радианах.

1. **Команды тригонометрические (угол задается в радианах):**

- **FPTAN** ; ST(1) = tg(ST), где -263 ≤ ST ≤ 264 и в ST – единица
- **FPATAN** ; ST = arctg(ST(1)/ST(0)), операнды из стека извлекает
- **FSIN** ***;*** ST = sin(ST), где -263 ≤ ST ≤ 264
- **FCOS** ***;*** ST = cos(ST), где -263 ≤ ST ≤ 264
- **FSINCOS** ***;*** ST(1) = sin(ST), где -263 ≤ ST ≤ 264 и ST= cos(ST)

2. Команды логарифмические и показательные (по основанию 2)

- **F2XM1** ; ST =2ST-1, где -1 ≤ ST ≤ 1 – обеспечивает более точные значения вблизи 1
- **FYL2X** ; ST =ST(1)*log2ST , где ST > 0
- **FYL2XP1** ; ST =ST(1)*log2(ST+1) , где –(1-2½/2)≤ ST ≤ 1-2½/2 – обеспечивает более точные значения вблизи 1

Оглавление

3.5 Команды управления сопроцессором

Группа команд управления предназначена для управления работой сопроцессора. Мнемоники управляющих команд могут начинаться с FN (без ожидания и без проверки особых случаев) или F (с ожиданием). Команды с буквой N в мнемокоде выполняются немедленно, что позволяет сэкономить несколько тактов.

- **FNSTCW (FSTCW)** ; записать содержимое CW в опер. память
- **FLDCW ;** загрузить CW из оперативной памяти
- **FNSTSW (FSTSW) ;** записать SW в оперативную память
- **FNSTSW AX (FSTSW AX) ;** записать CW в AX
- **FNCLEX (FCLEX) ;** сбросить флаги особых случаев в SW и биты ES, B
- **FNINIT (FINIT) ;** инициализировать сопроцессор:
  - управляющий регистр – бесконечность со знаком, округление к ближайшему, расширенная точность, все особые случаи замаскированы
  - регистр состояния – B=0 (бит занятости сброшен), код условия не определен, ST=ES=0, флаги особых случаев установлены в нуль
  - регистр тегов – все поля регистра тегов содержат значение 11 (пустой регистр))
- **FNSTENV (FSTENV)** ***;*** записать в память среду (содержимое всех регистров, кроме численных, в предопределенном формате***)***
- **FLDENV** ; загрузить среду
- **FNSAVE (FSAVE)** ; записать полное состояние (дополнительно сохраняет содержимое численных регистров)
- **FRSTOR** ; восстановить полное состояние
- **FINCSTP** ; увеличить указатель стека TOP на 1
- **FDECSTP** ; уменьшить указатель стека TOP на 1
- **FFREE** ; освободить регистр
- **FNOP** ; нет операции (не производит никаких действий)
- **FSETPM** ; установить защищенный режим работы (переводит сопроцессор в защищенный режим работы)

Оглавление

Контрольные вопросы

1. Поясните систему наименования команд сопроцессора.

Ответ.

2. Назовите основные команды передачи данных.

Ответ.

3. Перечислите основные арифметические команды. Поясните особенности их выполнения.

Ответ.

4. Расскажите, в каком виде фиксируется результат операции сравнения вещественных чисел. Как его использовать в основном процессоре?

Ответ.

5. Перечислите особенности использования трансцедентальных функций.

Ответ.

6. Перечислите наиболее важные возможности, реализуемые с использованием команд управления сопроцессором.

Ответ.

Оглавление

4 Методика разработки программ с использованием сопроцессора на ассемблере

Прежде, чем переходить к программированию сопроцессора, необходимо научиться работать с вещественными числами, переводить их из десятичного формата в двоичный и видеть эти числа в отладчике в шестнадцатеричном представлении. Для этого, нижеприведенный пример следует набрать в среде Radasm, откомпилировать, собрать и запустить в режиме отладки. Затем выполнить рекомендуемые в примере действия.

Пример 1. Работа с форматами вещественных чисел.

Написать программу сложения двух вещественных чисел 45.56 и 14.31. В сумме должны получить 59.87. Самостоятельно осуществить перевод слагаемых в короткий формат, а результат в короткий и длинный форматы. Просмотреть в отладчике внутренний формат чисел в памяти. Сравнить результат в памяти с результатом, полученным при переводе вручную. Изменить формат. Просмотреть результат. Изменить данные, пересоздать проект и выполнить программу с другими данными.

```
          .586
          .MODEL  flat, stdcall
          OPTION CASEMAP:NONE
Include kernel32.inc
Include masm32.inc
IncludeLib kernel32.lib
IncludeLib masm32.lib
          .CONST
MsgExit   DB    "Press Enter to Exit",0AH,0DH,0
nline     DB    13,10,0
          .DATA
a         dd       45.56 ;  42 36 3D 71   короткие форматы
b         dd       14.31 ;  41 64 F5 c3 или c2
          .DATA?
rezd      dd     ?   ; 42 6F 7A E2 или E1
```

Оглавление

```
rezq     dq     ?   ; 40 4D EF 5C 38   формат двойной точности
inbuf    DB    100 DUP (?)
outbuf   DB    20 Dup(?)
         .CODE
Start:   finit        ; инициализация сопроцессора
         fld a        ; загрузка в стек первого слагаемого
         fadd b       ; сложение числа в стеке со вторым слагаемым
         fst rezd     ;запись в память результата из стека  в коротком формате
         fstp rezq    ; запись в память результата с очисткой стека
                      ; в длинном формате.
; вывод результата на экран в формате чисел DQ
         Invoke FloatToStr, rezq, ADDR outbuf
         Invoke StdOut, ADDR outbuf
         Invoke StdOut, ADDR nline
         XOR    EAX,EAX
         Invoke StdOut,ADDR MsgExit
         Invoke StdIn,ADDR inbuf,LengthOf inbuf
         Invoke ExitProcess,0
         End    Start
```

На рисунке 2 представлен дамп памяти в отладчике на момент сохранения результата.

Оглавление

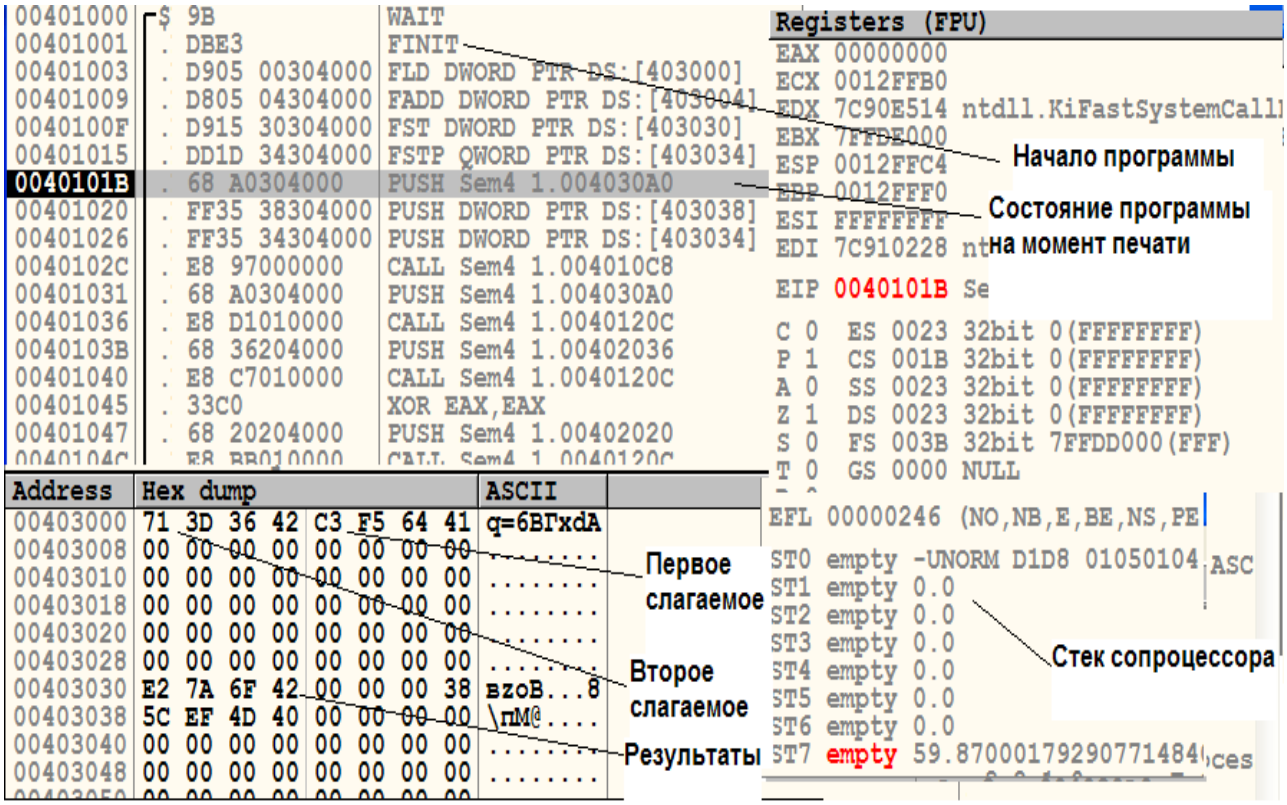

Рисунок 2 – Вид окна отладчика при пошаговом выполнении программы

Методика написания программ для сопроцессора имеет свои особенности. Главная причина этого в стековой организации сопроцессора. Чтобы написать программу для вычисления некоторого выражения, его нужно сначала представить в удобном для сопроцессора виде – в виде дерева разбора выражения. Листьями такого дерева являются операнды, а узлы соответствуют операциям. Затем его необходимо обойти слева направо, выполняя операции в узлах.

Пример 2. Пусть необходимо вычислить значение выражения (a-b)*c + (b-5)/d.

Строим дерево разбора выражения. Для этого определяется «главная» операция. В нашем выражении это операция «+». Именно она будет располагаться в корне дерева. Слева и справа расположатся, соответственно, левый и правый операнды. Операнды тоже являются выражениями. Поэтому для каждого из них необходимо построить свои поддеревья. В каждом из выражений определим главную операцию и поместим ее, соответственно, в корне левого и правого поддеревьев. Так, для левого поддерева такой операцией станет умножение, а в правом – деление. Левые операнды этих операций тоже являются выражениями. Для них тоже строим поддеревья. И в левой ветви выражение является разностью, и в правой. Поэтому, помещаем в корень поддеревьев операции «-». Теперь операнды – числовые значения. После это-

Оглавление

го процесс построения дерева завершен. На рисунке 3 представлено дерево, полученное в результате разбора дерева выражения.

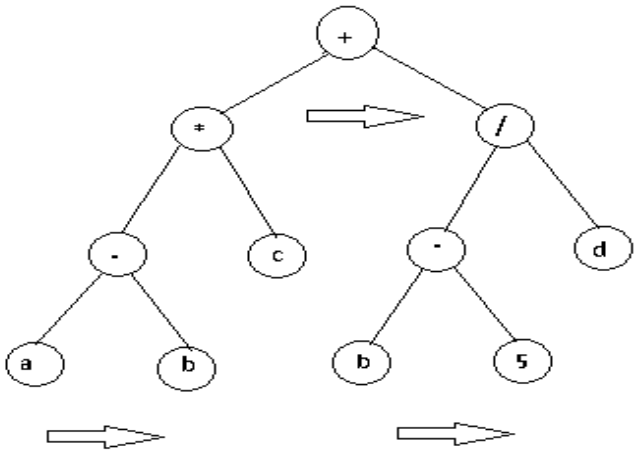

Рисунок 3 - Дерево разбора выражения

Обход дерево с крайней левой вершины определяет последовательность вычислений. При этом переход к корню выполняется только тогда, когда результаты всех нижних вершин посчитаны.

Алгоритм обработки дерева:

1. Если анализируемая вершина операнд – занести его значение в стек.
2. Если очередная вершина – операция – выполнить ее над одним или двумя операндами, определяемыми поддеревьями. Результат операции поместить в вершину стека.
3. Если в дереве еще есть вершины – перейти к шагу 1, иначе в вершине стека находится результат.

Однако при программировании следует учитывать ограниченность глубины стека, несовпадение форматов операндов и отсутствие команд для некоторых операций.

Текст программы приведен ниже.

```
        .586.
        .MODEL  flat, stdcall
        OPTION CASEMAP:NONE
Include kernel32.inc
Include masm32.inc
IncludeLib kernel32.lib
IncludeLib masm32.lib
```

Оглавление

```
          .CONST
MsgExit   DB    "Press Enter to Exit",0AH,0DH,0
nline     db     13,10,0
          .DATA
a         dd   10.5
b         dd    6.5
cc        dd    3.0
d         dd    2.0
con5      dd    5.0
          .DATA?
rez       dq    ?
inbuf     DB    100 DUP (?)
outbuf    db   20 dup(?)
          .CODE
Start:
        finit     ; инициализация сопроцессора
        fld a     ; занести в стек операнд a
        fld b     ; занести в стек операнд b
        fsubp st(1),st  ; вычесть из a b и результат занести в стек
        fld cc    ;  занести в стек операнд cc
        fmulp st(1),st ; умножить  разность a и b на cc и результат занести
                       ; в стек
        fld b     ;  занести в стек операнд b
        fld con5  ; занести в стек операнд con5
        fsubp st(1),st ; вычесть из  b con5  и результат занести в стек
        fld d     ;  занести в стек операнд con5
        fdivp st(1),st ; поделить разность b и con5 на d и результат занести
                       ; в стек
        fadd      ;  сложить результаты умножения и деления и результат
                       ; поместить в стек
        fstp rez  ; результат вычисления записать в память
```

Оглавление

```
invoke FloatToStr,rez,ADDR outbuf ; формирование и
Invoke StdOut,ADDR outbuf         ; вывод результата
Invoke StdOut,ADDR nline
XOR    EAX,EAX
Invoke StdOut,ADDR MsgExit
Invoke StdIn,ADDR inbuf,LengthOf inbuf
Invoke ExitProcess,0
End    Start
```

Однако если внимательно просмотреть особенности выполнения различных форматов команд, которые позволяют работать с операндами, размещенными в памяти, то полученную программу можно упростить, выполнив ту же последовательность вычислений, но меньшим количеством команд.

Ниже приведен фрагмент усовершенствованной программы вычисления:

```
finit
fld   a  ; занести в стек операнд a
fsub  b  ; вычесть из a b и результат занести в стек
fmul  cc ; умножить  разность a и b  на cc и результат занести в стек
fld   b  ; занести в стек операнд b
fsub con5 ; вычесть из  b con5  и результат занести в стек
fdiv  d  ; поделить  разность b и con5  на d и результат занести в стек
fadd  ; сложить результаты умножения и деления и результат поместить в стек
fstp rez  ; результат вычисления записать в память
```

Этот пример демонстрирует, что к решению любой задачи нужно подходить творчески. Прежде, чем программировать вычисление, необходимо внимательно просмотреть различные варианты реализации одного и того же алгоритма вычисления выражения.

Пример 3. Написать программу вычисления простых выражений

R=A*B; R=$(A+B)^2$; R=A+B*C+B/C; над операндами различных форматов.

```
        .586
        .MODEL  flat, stdcall
        OPTION CASEMAP:NONE
Include kernel32.inc
```

Оглавление

```
Include masm32.inc
IncludeLib kernel32.lib
IncludeLib masm32.lib
          .CONST
MsgExit   DB    13,10,"Press Enter to Exit",0AH,0DH,0
nline     DB    13,10,0
          .DATA
AI        WORD -5    ; переменные целого типа
BI        WORD 6
AR        DWORD 1.5      ; переменные вещественного типа
BR        DWORD -10.0
CR        DWORD 3.0
RI        DWORD ?
RD        DWORD ?
RQ        QWORD ?
RIQ       QWORD ?
          .DATA?
inbuf     DB    100 DUP (?)
outbuf    DB    20 DUP (?)
          .CODE
Start:
;  R=A*B = -30      вычисление выражения с целыми операндами
          finit
          fild   AI
          fimul  BI
          fistp  RI
          Invoke dwtoa, RI, ADDR outbuf
          Invoke StdOut, ADDR outbuf
          Invoke StdOut, ADDR nline
;  R=(A+B)^2 = 72.25   вычисление выражения с вещественными операндами
          fld    AR
          fadd   BR
```

Оглавление

```
        fmul   st,st
        fstp   RQ
        Invoke FloatToStr, RQ, ADDR outbuf
        Invoke StdOut, ADDR outbuf
        Invoke StdOut, ADDR nline
```

; R=A+B*C+B/C = -31,8333 вычисление выражения с вещественными операндами

```
        fld    AR
        fld    BR
        fmul   CR
        fadd
        fld    BR
        fdiv   CR
        fadd
        fstp   RQ
        Invoke FloatToStr, RQ, ADDR outbuf
        Invoke StdOut, ADDR outbuf
        XOR    EAX,EAX
        Invoke StdOut,ADDR MsgExit
        Invoke StdIn,ADDR inbuf,LengthOf inbuf
        Invoke ExitProcess,0
        End    Start
```

Пример 4. Написать программу вычисления выражений:

$z=(\sqrt{|x|-y})^2$; $len=\sqrt{(x^2+y^2)}$; с использованием дополнительных команд**.**

```
        .586
        .MODEL  flat,stdcall
        OPTION CASEMAP:NONE
Include kernel32.inc
Include masm32.inc
IncludeLib kernel32.lib
IncludeLib masm32.lib
         .CONST
MsgExit  DB    13,10,"Press Enter to Exit",0AH,0DH,0
```

Оглавление

```
nline    DB    13,10,0
         .DATA
x        dq    -29.1e-1
y        dd    4.6
z        dq    ?
len      dq    ?
;  z=(sqrt(|x|)-y)^2)=(1.7059-4.6)^2=8.375
         .DATA?
inbuf    DB    100 DUP (?)
outbuf   DB    20 DUP (?)
         .CODE
Start:
;        finit
         fld    x  ; st(0)=x
         fabs       ; st(0)=abs(x)
         fsqrt      ; sqrt(st(0))
         fsub   y   ; st(0)-y
         fst    st(1) ;st(1)=st(0)
         fmul          ; st(0)*st(1) -> sqr
         fstp z
         Invoke FloatToStr,z,ADDR outbuf
         Invoke StdOut, ADDR outbuf
         Invoke StdOut, ADDR nline
;  len=sqrt(x^2+y^2)= sqrt(2.91^2 +4.6^2)=5.4431
         fld    x
         fmul   st,st
         fstp   len
         fld    y
         fmul   st,st
         fadd   len
         fsqrt
```

Оглавление

```
        fstp   len
        Invoke FloatToStr, len, ADDR outbuf
        Invoke StdOut, ADDR outbuf
        Invoke StdOut, ADDR nline
        XOR    EAX,EAX
        Invoke StdOut,ADDR MsgExit
        Invoke StdIn,ADDR inbuf,LengthOf inbuf
        Invoke ExitProcess,0
        End    Start
```

Пример 5. Написать программу вычисления суммы ряда

$$Y = \sum_{i=1}^{10} 1/i! \text{ , где } i!=1*2*3*\ldots.*i.$$

Особенностью задачи является накопление в цикле значения суммы ряда и вычисление факториала очередного числа через предыдущее значение. Накопленную текущую сумму будем хранить в памяти, а все данные для формирования факториала и очередного члена ряда будем хранить в стеке. Ниже приведен текст программы.

```
; y=1+1/2!+1/3!+.....+ 1/10!
        .586
        .MODEL  flat, stdcall
        OPTION CASEMAP:NONE
Include kernel32.inc
Include masm32.inc
IncludeLib kernel32.lib
IncludeLib masm32.lib
        .CONST
MsgExit  DB    13,10,"Press Enter to Exit",0AH,0DH,0
        .DATA
i        equ   10
y        dq    0
        .DATA?
inbuf    DB    100 DUP (?)
outbuf   DB    20 dup(?)
```

Оглавление

```
          .CODE
Start:    xor   ecx,ecx     ; очистка регистра счетчика
          fld1              ; st(0)=1
          fld1              ; st(0)=i=1  st(1)=1!
          fst   y           ; сохранение первого члена суммы
          mov  cx,i-1       ;  установка счетчика циклов на 9
cycl:     fld1              ; st(0)=1
          fadd              ; 1+ st(1) -> st(1)=i=2,3,….
          fmul   st(1),st(0) ; st(0)=i=2,3….. st(1)=i!
          fld1               ; st(0)=1
          fdiv st(0),st(2) ;  1/i!    st(0)=1  st(2)= i!
          fadd y ;  накопление суммы ряда – добавление очередного члена ряда
          fstp y  ; сохранение промежуточного значения суммы ряда
          loop cycl
          invoke FloatToStr,y,ADDR outbuf
          invoke StdOut,ADDR outbuf
          XOR    EAX,EAX
          Invoke StdOut,ADDR MsgExit
          Invoke StdIn,ADDR inbuf,LengthOf inbuf
          Invoke ExitProcess,0
          End    Start
```

Пример 6. Вычислить с точностью ε сумму бесконечного ряда

$$S = X + \frac{X^3}{Y^3 + X} + \frac{X^5}{Y^6 + X^2} + \ldots + \frac{X^{2N+1}}{Y^{3N} + X^N} + \ldots .$$

Особенностью этой задачи является вычисление суммы ряда с определенной точностью. А это значит, что использовать счетный цикл нельзя и для продолжения вычислений необходимо проверять очередной член ряда. Если он меньше указанной точности, то сумма посчитана, и нужно выводить результат. В противном случае необходимо выполнить переход на повторение вычислений. Эта операция выполняется с помощью команд сравнения и условных переходов. Как отмечалось ранее, анализ кодов условий выполняет основной процессор,

поэтому ему нужно передать результаты, полученные в операциях сравнения. Кроме того, из формулы видно, что и числитель и знаменатель можно считать отдельно и их значения зависят от значений предыдущих вычислений. Поэтому можно использовать рекуррентные соотношения и промежуточные результаты хранить в памяти.

```
            .586
            .MODEL  flat, stdcall
            OPTION CASEMAP:NONE
Include kernel32.inc
Include masm32.inc
IncludeLib kernel32.lib
IncludeLib masm32.lib
; s=x+x^3/(y^3+x)+x^5/(y^6+x^2)+x^7/(y^9+x^3).....
            .CONST
MsgExit  DB    13,10,"Press Enter to Exit",0AH,0DH,0
Msgs     db    13,10,"summa= ",0
Msgsr    db    13,10,"elemet ryada =  ",0
            .DATA
eps      dq    0.001
x        dq    3.0
y        dq    5.0
x2n      dq    0  ; ячейка для накопления X^(2N-1)
y3       dq    0 ; ячейка для хранения Y^3
yn       dq    1.0 ; ячейка для накопления Y^3N
xn       dq    1.0 ; ячейка для накопления X^N
r        dq    ?  ; текущий член ряда
s        dq    0  ; сумма ряда
            .DATA?
inbuf    DB    100 DUP (?)
outbuf   DB    100 dup(?),13,10,0
            .CODE
Start:   xor    eax,eax
```

Оглавление

```
        fld    x        ; st(0)=x
        fst    s
        fstp   x2n
        fld    y
        fmul   y
        fmul   y
        fstp   y3
cycl:   fld    yn
        fmul   y3
        fst    yn
        fld    x
        fmul   xn
        fst    xn
        fadd
        fld    x
        fmul   x
        fmul   x2n
        fst    x2n
        fdivr
        fst    r
        fadd   s
        fstp   s
        Invoke StdOut,ADDR Msgsr
        invoke FloatToStr,r,ADDR outbuf
        invoke StdOut,ADDR outbuf
        fld    eps
        fcomp  r
        fstsw  ax
        sahf
        jc     cycl
        Invoke StdOut,ADDR Msgs
        Invoke FloatToStr,s,ADDR outbuf
```

Оглавление

```
Invoke StdOut,ADDR outbuf
XOR    EAX,EAX
Invoke StdOut,ADDR MsgExit
Invoke StdIn,ADDR inbuf,LengthOf inbuf
Invoke ExitProcess,0
End    Start
```

Следует иметь в виду, что при программировании таких задач очень велик риск переполнения стека. Причем никаких сообщений об этом сопроцессор не формирует. Просто программа начинает выдавать неверный результат. Для обнаружения ошибки необходимо при отладке внимательно выполнить программу, отслеживая наличие вершины стека и данные управляющих регистров. Если вершина стека исчезнет, а процесс еще не завершен, следует искать ошибку. Чаще всего она связана с отсутствием извлечения ненужной информации из стека в процессе вычислений.

Оглавление

Контрольные вопросы

1. Сформулируйте последовательность представления сложных выражений в виде двоичного дерева. Как такое дерево реализуется в программе?

Ответ.

2. Сформулируйте методику разработки программ с использованием вещественной арифметики.

Ответ.

Оглавление

Литература

1. Ирвин К. Язык ассемблера для процессоров Intel. – М.: Издательский дом «Вильямс», 2005.
2. Юров В.И. Справочник по языку Ассемблера IBM PC. – СПб.: Питер, 2002.

Оглавление