

ПРАКТИЧЕСКОЕ ЗАНАТИЕ_2

ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ

Цель работы:

Практическое овладение навыками разработки программного кода на языке Ассемблер. Изучение внутреннего представления целых и вещественных чисел. Практическое освоение основных функций отладчика TD.

Задачи:

1. Выполнить перевод заданных преподавателем целых чисел из десятичной в двоичную и шестнадцатеричную систему счисления. Дать их внутреннее (машинное) представление в соответствии с диапазоном в знаковых и беззнаковых форматах типов ShortInt (signed char), Byte (unsigned char), Integer (int), Word (unsigned int).
2. Выполнить перевод заданных преподавателем вещественных чисел из десятичной в двоичную и шестнадцатеричную систему счисления. Дать их внутреннее (машинное) представление в двоичной и шестнадцатеричной системах счисления в форматах типов Single (float), Double (double), Extended (long double).

Порядок выполнения работы:

1. Вычислить для своего варианта [целые числа](#);
2. Перевести все числа из десятичной в двоичную систему и шестнадцатеричную систему счисления (Файл Представление чисел);
3. Получить их внутреннее представление в соответствии с диапазоном и типом;
4. Написать программу описания этих чисел на языке Ассемблера и получить листинг;
5. Проверить правильность своих выкладок, используя отладчик.
6. Проанализировать [дамп памяти](#).
7. Вычислить для своего варианта [вещественные числа](#);
8. Перевести все числа из десятичной в двоичную систему счисления (Файл Представление чисел);
9. Получить их внутреннее представление в соответствии с диапазоном и типом;
10. Написать программу описания этих чисел на языке Ассемблера. Создать загрузочный модуль, получить листинг;
11. Проверить правильность своих выкладок, используя отладчик.

Теоретическая часть

Директивы определения данных используются для выделения байт в сегменте. При написании программы они также используются для заполнения этих байт начальными данными и определения переменных данных.

Все директивы определения данных имеют некоторые общие средства. Во первых они могут генерировать инициализированные данные и резервировать место для неинициализированных данных. Для инициализированных данных определяется некоторое начальное значение. Неинициализированные данные определяются без задания начального значения (говорят, что их начальное значение является неопределенным). В директивах определения данных неинициализированные данные указываются с помощью символа ?. Все прочее должно представлять значение инициализированных данных.

В ассемблере TASM необходимо размещать инициализированные данные в сегменте .data.

Рассмотрим основной формат определения данных:

[имя] **Dn** *выражение*

Имя элемента данных не обязательно (это указывается квадратными скобками), но если в программе имеются ссылки на некоторый элемент, то это делается посредством имени.

Для определения элементов данных имеются следующие директивы: **DB** (байт), **DW** (слово), **DD** (двойное слово), **DQ** (четверное слово) и **DT** (десять байт).

В директиве **db** (*байт*) можно задавать следующие значения:

- выражение-константу, имеющую значения в диапазоне от –128 до 255 (байты со знаком в диапазоне от –128 до +127; беззнаковые байтовые значения в диапазоне от 0 до 255);
- 8-битовое относительное выражение, использующее операции

high (для выделения старшего байта в слове) и *low* (для выделения младшего байта в слове);

- символьную строку из одного или более символов с использованием стандартного формата заключенной в кавычки строки. В этом случае определяется несколько байт, по одному на каждый символ строки.

Значениями директивы **dw** (*слово*) могут быть:

- выражение-константа в диапазоне от –32768 до 65535 (слова со знаком в диапазоне от –32768 до 32767, беззнаковые слова в диапазоне от 0 до 65535);
- относительное выражение, занимающее 16 или менее бит (включая смещение в 16-битовом сегменте, сегмент или значение группы);
- одно- или двухбайтовая строка в стандартном формате (строка, заключенная в кавычки).

Значениями директивы ***dd*** (*двойное слово*) могут быть:

- выражение-константа в диапазоне от -2147483648 до 4294967295 (при выборе процессора i80386 и выше) или от -32768 до 65535 в противном случае;
- относительное адресное выражение, состоящее из 16-битового сегмента и 16-битового смещения;

строка длиной до 4 символов в стандартном формате (строка, заключенная в кавычки).

Значениями директивы ***dq*** (*четверное слово*) могут быть:

- выражение-константа в диапазоне от -2147483648 до 4294967295 (при выборе процессора i80386 и выше) или от -32768 до 65535 в противном случае;
- относительное или адресное выражение, состоящее из 32 или менее бит (при выборе процессора i80386 и выше) или 16 или менее бит (для всех других процессоров);
- положительная или отрицательная константа, имеющая значение в диапазоне от -2^{63} до $2^{64} - 1$ (четверное слово со знаком в диапазоне от -2^{63} до $2^{63} - 1$, беззнаковое четверное слово в диапазоне от 0 до $2^{64} - 1$);
- строка длиной до 8 байт в стандартном формате (строка, заключенная в кавычки).

Значениями директив ***df*** и ***dp*** (48-битовый дальний указатель процессора 80386) могут быть:

- выражение-константа в диапазоне от -2147483648 до 4294967295 (при выборе процессора i80386 и выше) или от -32768 до 65535 в противном случае;
- относительное или адресное выражение, состоящее из 32 или менее бит (при выборе процессора i80386 и выше) или 16 или менее бит (для всех других процессоров);
- относительное адресное выражение, состоящее из 16-битового сегмента и 32-битового смещения;
- положительная или отрицательная константа, имеющая значение в диапазоне от -2^{47} до $2^{48} - 1$ (четверное слово со знаком в диапазоне от -2^{47} до $2^{47} - 1$, беззнаковое четверное слово в диапазоне от 0 до $2^{48} - 1$);
- строка длиной до 6 байт в стандартном формате (строка, заключенная в кавычки).

Значениями директивы ***dt*** могут быть:

- выражение-константа в диапазоне от -2147483648 до 4294967295 (при выборе процессора i80386 и выше) или от -32768 до 65535 в противном случае;
- относительное или адресное выражение, состоящее из 32 или менее бит (при выборе процессора i80386 и выше) или 16 или менее бит (для всех других процессоров);
- относительное адресное выражение, состоящее из 16-битового сегмента и 32-битового смещения;

- положительная или отрицательная константа, имеющая значение в диапазоне от -2^{79} до $2^{80} - 1$ (четверное слово со знаком в диапазоне от -2^{79} до $2^{79} - 1$, беззнаковое четверное слово в диапазоне от 0 до $2^{80} - 1$); - строка длиной до 10 байт в стандартном формате (строка, заключенная в кавычки);
- упакованная десятичная константа, имеющая значение в диапазоне 48 от 0 до 99999999999999999999.

При сохранении данных в памяти в программной модели IA-32 младшее значение всегда записывается перед старшим значением.

Выражение может содержать константу, например

FLD1 db 25

или знак вопроса для неопределенного значения, например

FLDB db ?

Выражение может содержать несколько констант, разделенных запятыми и ограниченными только длиной строки:

FLD3 db 11, 12, 13, 14, 15, 16, ...

Ассемблер определяет эти константы в виде последовательности смежных байт. Ссылка по имени FLD3 указывает на первую константу, 11, по FLD3+1 - на вторую, 12. (FLD3 можно представить как FLD3+0). Например, команда `MOV AL,FLD3+3` загружает в регистр AL значение 14 (шест. 0E).

Выражение допускает также повторение константы в следующем формате:

имя директива [выражение_dup [выражение_dup]

TASM инициализирует «имя» таким образом, чтобы оно указывало на резервируемую директивой область. Тип данной переменной зависит от фактически используемой директивы.

Каждое «*выражение_dup*» может иметь следующий синтаксис:

- ?;
- значение;
- счетчик dup (*выражение_dup*[, *выражение_dup*]), где «счетчик» задает, сколько раз будет повторяться блок данных;

«счетчик» не может быть относительным и иметь опережающие ссылки.

Если вы хотите получить неинициализированные данные, используйте идентификатор ?. Объем резервируемой для неинициализированных данных памяти зависит от фактически используемой директивы.

Выражение «значение» предназначено для фактического описания отдельного элемента данных в каждой директиве. В некоторых директивах поле значения может быть очень

сложным и содержать много элементов, другие могут потребовать только простого выражения.

Следующие три примера иллюстрируют повторение:

A1 dw 10 *dup* (?) ;Десять неопределенных слов
A2 db 5 *dup* (14) ;Пять байт, содержащих шест.14
A3 db 3 *dup* (4 DUP(8)) ;Двенадцать восьмерок

В следующем примере используется директива dw, которая выделяет слова:

var1 dw 2 *dup* (3 *dup* (1,3), 5)
что эквивалентно директиве
var1 dw 1,3,1,3,1,3,5,1,3,1,3,1,3,5

Синтаксис поля «значение» для каждой из этих директив различается и основывается на возможности представлять отдельные величины с помощью данных определенного размера (например, байтовые данные нельзя интерпретировать, как число с плавающей точкой).

Выражение может содержать символьную строку или числовую константу. Символьная строка используется для описания данных, таких как, например, имена людей или заголовки страниц.

Содержимое строки отмечается одиночными кавычками, например, 'PC' или двойными кавычками - "PC". Ассемблер переводит символьные строки в объектный код в обычном формате ASCII.

Символьная строка определяется только директивой DB, в которой указывается более двух символов в нормальной последовательности слева направо. Следовательно, директива *db* представляет единственно возможный формат для определения символьных данных.

Числовые константы используются для определения арифметических величин и адресов памяти. Для описания константы кавычки не ставятся. Ассемблер преобразует все числовые константы в шестнадцатеричные и записывает байты в объектном коде в обратной последовательности - справа налево.

При записи символьных и числовых констант следует помнить, что, например, символьная константа, определенная как DB '12', представляет символы ASCII и генерирует шест. 3132, а числовая константа, определенная как DB 12, представляет двоичное число и генерирует шест. 0C

Таблица примеров определения переменных в программе на ассемблере.

Имя	Директива определения	значения	комментарий
DB – определение байтов			
BYTE1	db	?	Без указания значения
BYTE2	db	48	Десятичная константа

BYTE3	db	30H	Шестнадцатеричная константа
BYTE4	db	01111010B	Двоичная константа
BYTE5	db	10 DUP(0)	Десять нулевых значений
BYTE6	db	'Input symbol'	Строка символов
BYTE7	db	'12345'	Строка цифровых символов
BYTE8	db	01, 'Jan', 02, 'Feb', 03, 'mar', 04,	Таблица месяцев года, состоит из числовых значений номера месяца и его сокращенного названия
DW – определение слов (16 бит=2 байта)			
WORD1	dw	0fff0h	Шестнадцатеричная константа
WORD2	dw	01111010B	Двоичная константа
WORD3	dw	BYTE3	Константа адреса переменной
WORD4	dw	2,3,4,5,6	Таблица из 5 констант
WORD5	dw	8 DUP(0)	Восемь нулевых значений
DD – определение двойных слов (32 бита = 4 байта)			
DWORD1	dd	?	Без значения
DWORD2	dd	41562	Десятичное значение
DWORD3	dd	48H, 24H, CA	3 шестнадцатеричные константы
DWORD4	dd	WORD1	Адрес слова в виде сегмент:смещение
DWORD5	dd	BYTE8-BYTE5	Разность между адресами переменных (количество байт)

В таблице показаны различные числовые форматы.

Десятичный формат. Десятичный формат допускает десятичные цифры от 0 до 9 и обозначается последней буквой D, которую можно не указывать, например, 125 или 125D. Несмотря на то, что ассемблер позволяет кодирование в десятичном формате, он преобразует эти значения в шест. объектный код.

Например, десятичное число 125 преобразуется в шест. 7D.

Шестнадцатеричный формат. Этот формат допускает шест. цифры от 0 до F и обозначается последней буквой H. Так как ассемблер полагает, что с буквы начинаются идентификаторы, то первой цифрой шест. константы должна быть цифра от 0 до 9. Например, 2EH или 0FFFH, которые ассемблер преобразует соответственно в 2E и FF0F (байты во втором примере записываются в объектный код в обратной последовательности).

Двоичный формат. Двоичный формат допускает двоичные цифры 0 и 1 и обозначается последней буквой B. Двоичный формат обычно используется для более четкого представления битовых значений в логических командах AND, OR, XOR и TEST.

Десятичное число 12, шест. C и двоичное 1100B все генерируют один и тот же код: шест. 0C или двоичное 0000 1100 в зависимости от того, как вы рассматриваете содержимое байта.

Восьмеричный формат. Восьмеричный формат допускает восьмеричные цифры от 0 до 7 и обозначается последней буквой Q или O, например, 253Q. На сегодня восьмеричный формат используется весьма редко.

Отображение данных в памяти

Пусть сегмент данных содержит нижеприведенные переменные, первая из которых записана в самом начале сегмента.

Смещение переменной	Идентификатор (имя) переменной	Описание переменной	Комментарий
0000	CHAR	DB ?	Неинициализированный байт
0001	SOURCE	DB 10,20,30	Вектор десятичных значений
0004	DEST	DW 3 DUP(?)	Зарезервированные слова
000A	MESS	DB 'Name? , \$'	Текст запроса
0010	REZ	DD ?	Зарезервированное двойное слово

Если Вы будете просматривать дамп памяти с такой информацией, то увидите примерно такую картину:

0000	00	0A	14	1E	00	00	00	00	
0008	00	00	4E	61	4D	45	3F	20	Name?
0010	00	00	00	00					

(Цветное выделение сделано для пояснений!)

Здесь голубым цветом выделены поля значений смещения, розовым цветом – поля значений переменных и зарезервированных слов или байтов в шестнадцатеричном виде, зеленым цветом – те же поля переменных, но в символьном виде (Числовые значения, такие как 20 десятичное, равное 14 шестнадцатеричному, в символьном виде либо отображаются нечисловыми и небуквенными символами, либо не отображаются вовсе. В примере эти символы заменены подчеркиванием _).

Из приведенного примера очевидно, что память для переменных выделяется в соответствии с их описанием в сегменте данных: в том порядке, как они описываются, и столько байтов, каков размер переменных.

Пример решения типового варианта для целых чисел

Даны базовые числа: $X=\pm 2143, Y=\pm 30$.

Прибавить и отнять от этих чисел № своего варианта, таким образом, получив числа для перевода в двоичную и шестнадцатеричную систему счисления, а именно:

$$\begin{array}{ll} 30 + 50 = 80; & -30 + 50 = 20; \\ 30 - 50 = -20; & -30 - 50 = -80. \\ \\ 2143 + 50 = 2193; & -2143 + 50 = -2093; \\ 2143 - 50 = 2093; & -2143 - 50 = -2193; \end{array}$$

Машинное представление данных чисел в двоичной системе счисления с учётом типа и диапазона:

$$\begin{array}{ll} 80d \rightarrow 0101\ 0000b \text{ (Byte);} & -80d \rightarrow 1011\ 0000b; \\ 20d \rightarrow 0001\ 0100b \text{ (Byte);} & -20d \rightarrow 1110\ 1100b; \\ 2193d \rightarrow 0000\ 1000\ 1001\ 0001b \text{ (Word);} & -2193d \rightarrow 1111\ 0111\ 0110\ 1111b; \\ 2093d \rightarrow 0000\ 1000\ 0010\ 1101b \text{ (Word);} & -2093d \rightarrow 1111\ 0111\ 1101\ 0011b. \end{array}$$

Машинное представление данных чисел в шестнадцатеричной системе счисления с учётом типа и диапазона:

Byte	20d	→ 14h;	80d	→ 50h;
Shortint	- 80d	→ B0b;	- 20d	→ ECh;
	20d	→ 14h;	80d	→ 50h;
Word	20d	→ 0014h;	80d	→ 0050h;
	2093d	→ 082Dh;	2193d	→ 0891h;
Integer	- 2193d	→ F76Fh;	- 2093d	→ F7D3h;
	- 80d	→ FFB0b;	- 20d	→ FFECh;
	20d	→ 0014h;	80d	→ 0050h;
	2093d	→ 082Dh;	2193d	→ 0891h;
Longint	- 2193d	→ FFFF F76Fh;	2193d	→ 0000 0891h.

Набрать текст программы:

```
.MODEL small
.STACK 100h
.DATA
;---byte---
i db 20,80
;---shortint---
is db -80,-20
db 20,80
;---word---
iw dw 20,80
dw 2093
dw 2193
;---integer---
ii dw -2193
dw -2093
```

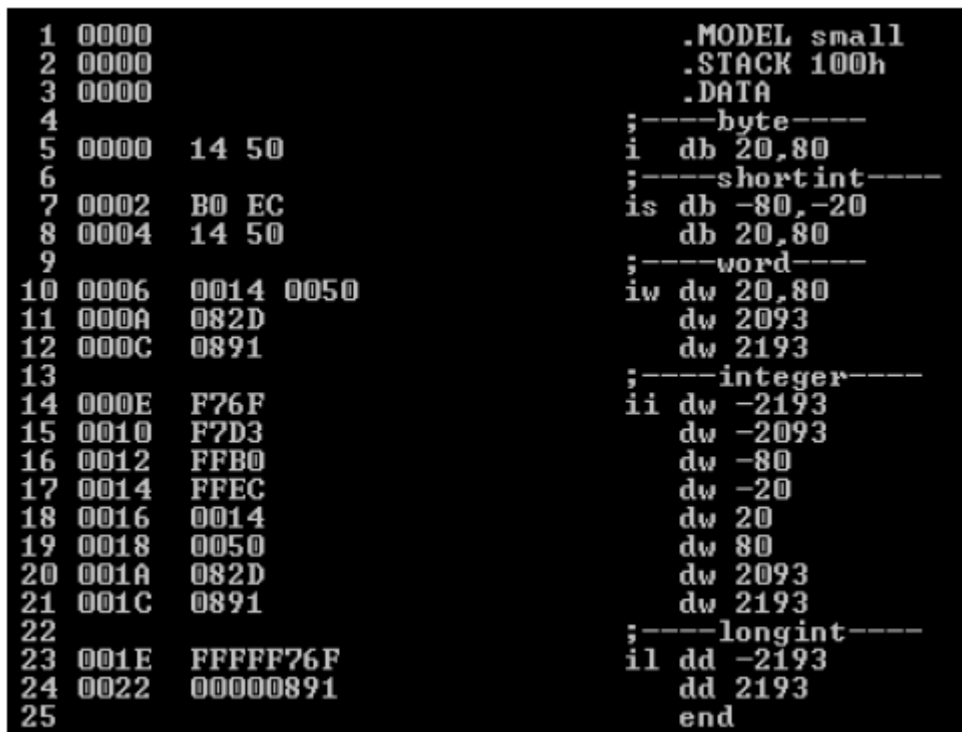


```

dw -80
dw -20
dw 20
dw 80
dw 2093
dw 2193
;----longint----
il dd -2193
dd 2193
...
end

```

Сохранить файл с именем `dig1.asm` (обратите внимание на расширение программы Ассемблер). Для получения результата откомпилируем программу: в командной строке MS-DOS наберите `tasm.exe dig1.asm/L`. Для просмотра появившегося листинга наберите команду `type dig1.lst`. Нас интересует часть содержимого, изображенная на рис. 1.



```

1 0000          .MODEL small
2 0000          .STACK 100h
3 0000          .DATA
4              ;----byte----
5 0000 14 50     i db 20,80
6              ;----shortint----
7 0002 B0 EC     is db -80,-20
8 0004 14 50     db 20,80
9              ;----word----
10 0006 0014 0050 iw dw 20,80
11 000A 082D     dw 2093
12 000C 0891     dw 2193
13              ;----integer----
14 000E F76F     ii dw -2193
15 0010 F7D3     dw -2093
16 0012 FF80     dw -80
17 0014 FFEC     dw -20
18 0016 0014     dw 20
19 0018 0050     dw 80
20 001A 082D     dw 2093
21 001C 0891     dw 2193
22              ;----longint----
23 001E FFFFFFF76F il dd -2193
24 0022 00000891 dd 2193
25              end

```

Рис. 1. Фрагмент файла листинга `dig1.lst`

Анализ этого файла показывает, что машинный формат целых чисел был получен правильно.

Пример решения типового варианта для вещественных чисел

Дано базовое число $\pm X.Y$. Прибавить и отнять от него номер своего варианта (для целой и дробной части отдельно).

Например, $X=\pm 81, Y=\pm 1432$, № 50. Получим вещественные числа:

<p>1) $\begin{array}{r} 81.1432 \\ + 50 \\ \hline 131.1482 \end{array}$</p>	<p>2) $\begin{array}{r} 81.1432 \\ - 50 \\ \hline 31.1432 \end{array}$</p>
<p>3) $\begin{array}{r} -81.1432 \\ + 50 \\ \hline -31.1432 \end{array}$</p>	<p>4) $\begin{array}{r} -81.1432 \\ - 50 \\ \hline -131.1432 \end{array}$</p>

Перевод десятичных чисел в двоичную систему счисления:

$31d \rightarrow 0001\ 1111b;$
 $0.1382d \rightarrow 0.00100011011000010001001101b;$
 $31.1382d \rightarrow 0001\ 1111.00100011011000010001001101b;$
 $131d \rightarrow 1000\ 0011b;$
 $0.1482d \rightarrow \dots$
 $131.1482d \rightarrow \dots$

Внутреннее представление данных чисел в шестнадцатеричной системе счисления с учётом типа и диапазона:

Dword	$-31d \rightarrow C1F80000h;$ $-0.1382d \rightarrow BE0D844Dh;$ $-31.1382d \rightarrow C1F91B09h;$ \dots		$31d \rightarrow 41F80000h;$ $0.1382d \rightarrow 3E0D844Dh;$ $31.1382d \rightarrow 41F91B09h;$
Qword	$-31d \rightarrow C03F000000000000h;$ $-0.1382d \rightarrow BFC1B089A0\dots h;$ $-31.1382d \rightarrow C03F236113\dots h;$ \dots		$31d \rightarrow 403F000000000000h;$ $0.1382d \rightarrow 3FC1B089A0\dots h;$ $31.1382d \rightarrow 403F236113\dots h;$
Long double	$-31d \rightarrow C003F80000000000000000h;$ $31d \rightarrow 4003F80000000000000000h;$ $-0.1382d \rightarrow BFFC8D844D0\dots h;$ $0.1382d \rightarrow 3FFC8D844D0\dots h;$ $-31.1382d \rightarrow C003F91B089\dots h;$ $31.1382d \rightarrow 4003F91B089\dots h;$		

Для проверки перевода в шестнадцатеричную систему счисления создадим листинг digt2.lst, используя следующий код

```
.MODEL small
.STACK 100h
.DATA
;----float (DWord)----
f dd -131.1482
dd 131.1482
dd -31.1382
```

```

dd 31.1382
dd 131.
dd -131.
dd 31.
17
dd -31.
dd 0.1482
dd -0.1482
dd 0.1382
dd -0.1382
;----double (QWord)----
d dq -131.1482
dq 131.1482
dq -31.1382
dq 31.1382
dq 131.
dq -131.
dq 31.
dq -31.
dq 0.1482
dq -0.1482
dq 0.1382
dq -0.1382
;----long double (Tbyte)----
t dt -131.1482
dt 131.1482
dt -31.1382
dt 31.1382
dt 131.
dt -131.
dt 31.
dt -31.
dt 0.1482
dt -0.1482
dt 0.1382
dt -0.1382
end

```

Нас интересует часть содержимого листинга, изображенная на рисунке. Анализ этого файла показывает, что машинный формат вещественных чисел был получен правильно¹.

¹ Заметим, что для некоторых чисел с дробной частью в форматах 64 и 80 бит за счет округления может получиться другой результат в младшей HEX – цифре. Кроме того, 16-разрядный TASM (tasm.exe) не поддерживает длинных имен, т.е. имен с количеством символов более чем восемь. 32-разрядный TASM (tasm32.exe) компилирует с тем же результатом, что и tasm.exe, но с поддержкой длинных имен.

```

1 0000 .MODEL small
2 0000 .STACK 100h
3 0000 .DATA
4 ;----float (DWord)----
5 0000 C30325F0 f dd -131.1482
6 0004 430325F0 dd 131.1482
7 0008 C1F91B09 dd -31.1382
8 000C 41F91B09 dd 31.1382
9 0010 43030000 dd 131.
10 0014 C3030000 dd -131.
11 0018 41F80000 dd 31.
12 001C C1F80000 dd -31.
13 0020 3E17C1BE dd 0.1482
14 0024 BE17C1BE dd -0.1482
15 0028 3E0D844D dd 0.1382
16 002C BE0D844D dd -0.1382
17 ;----double (QWord)----
18 0030 C06064BE0DED288D d dq -131.1482
19 0038 406064BE0DED288D dq 131.1482
20 0040 C03F236113404EA4 dq -31.1382
21 0048 403F236113404EA4 dq 31.1382
22 0050 4060600000000000 dq 131.
23 0058 C060600000000000 dq -131.
24 0060 403F000000000000 dq 31.
25 0068 C03F000000000000 dq -31.
26 0070 3FC2F837B4A2339C dq 0.1482
27 0078 BFC2F837B4A2339C dq -0.1482
28 0080 3FC1B089A0275254 dq 0.1382
29 0088 BFC1B089A0275254 dq -0.1382
30 ;----long double (Tbyte)----
31 0090 C0068325F06F69446738 t dt -131.1482
32 009A 40068325F06F69446738 dt 131.1482
33 00A4 C003F91B089A02752546 dt -31.1382
34 00AE 4003F91B089A02752546 dt 31.1382
35 00B8 40068300000000000000 dt 131.
36 00C2 C0068300000000000000 dt -131.
37 00CC 4003F800000000000000 dt 31.
38 00D6 C003F800000000000000 dt -31.
39 00E0 3FFC97C1BDA5119CE076 dt 0.1482
40 00EA BFFC97C1BDA5119CE076 dt -0.1482
41 00F4 3FFC8D844D013A92A305 dt 0.1382
42 00FE BFFC8D844D013A92A305 dt -0.1382
43 end

```