ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. А.Н. Тихонова  
Департамент компьютерной инженерии

Курс: Вычислительные системы и компьютерные сети

Отчёт   
по практической работе №3 «Intel-MIPS»

Студент: Омаров Марат Тимурович  
Группа: БИВ206  
Вариант: 8  
Дата: 06.10.2022

Москва 2022

Оглавление

[Условие для перевода 3](#_Toc115941066)

[Перевод числа в формат IEEE754-2008 3](#_Toc115941067)

[Решение 3](#_Toc115941068)

[Проверка результатов 5](#_Toc115941069)

[Условие задачи 5](#_Toc115941070)

[Код Intel 6](#_Toc115941071)

[Ссылка на github с кодом Intel 6](#_Toc115941072)

[Код 6](#_Toc115941073)

[Логика кода 7](#_Toc115941074)

[Разбор инструкций 8](#_Toc115941075)

[Запись переменной в вершину стека 8](#_Toc115941076)

[Использование FSIN 10](#_Toc115941077)

[Добавление новой переменной в стек 10](#_Toc115941078)

[Команда вычитания FSUB 11](#_Toc115941079)

[Команда сложения FADD 11](#_Toc115941080)

[Использование FST для записи ST 12](#_Toc115941081)

[Инструкция сравнения FCOMI 14](#_Toc115941082)

[Результат выполнения программы 14](#_Toc115941083)

[Код MIPS 15](#_Toc115941084)

[Ссылка на github с кодом MIPS 15](#_Toc115941085)

[Код 15](#_Toc115941086)

[Результат выполнения программы 19](#_Toc115941087)

[Разбор инструкций 19](#_Toc115941088)

[Разбор основных инструкций 19](#_Toc115941089)

# Условие для перевода

Дата моего дня рождения: 21.07.2003

Поскольку является нечётным, число A будет положительным.

Число

# Перевод числа в формат IEEE754-2008

## Решение

Будем производить перевод в тип одинарной точности **binary32 (float)**. Это значит, что наше число будет записано с характеристиками следующей разрядности: s = 1 (знак), p = 8 (порядок), m = 23 (мантисса).

1. Для начала переведём число A в двоичную СС.

Целая часть:

Для подсчёта использовался онлайн калькулятор (см. рис. 1, 2).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок . Подсчёт целой части числа А.

Дробная часть:

Таким образом:

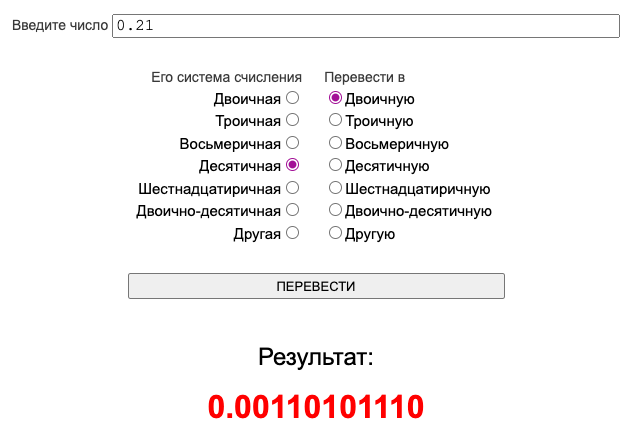


Рисунок . Подсчёт дробной части числа А.

1. Запишем число в нормализованном виде:
2. Определим мантиссу:

Для типа float необходимо выделить 23 + 1 + 1 бита (“1” целую часть не учитываем, а конец числа округляем). Значит,

25 разрядов числа

23 разряда мантиссы без старшей “1”.

Т.е., мантисса без старшей “1” составляет:

1. Определим порядок:
2. Результат

C

9

3

D

C

4

8

4

Таким образом:

## Проверка результатов

Для проверки воспользуемся инструментом из программы MIPS (см. рис. 3)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок . Проверка результатов перевода в MIPS.

Как видно из рисунка, результат полностью совпал.

# Условие задачи

Задать два числа A и B формата ЧПЗ:

* число A, полученное ранее;
* произвольное число B.

Сравнить число A с числом .

В Intel использовать fsin.

# Код Intel

## Ссылка на github с кодом Intel

На случай, если приведённый ниже код будет отображён некорректно, прилагаю ссылку на свой репозиторий, на который я выложил весь программный код:

<https://github.com/marato-o/CS-CN/blob/master/lab_3/Intel_lab_3.asm>

## Код

format PE64 Console 5.0

entry Start

include 'win64a.inc'

section '.bss' readable writeable

readBuf db ?

section '.idata' import data readable

library kernel,'KERNEL32.DLL'

import kernel, SetConsoleTitleA, 'SetConsoleTitleA', \

GetStdHandle, 'GetStdHandle', WriteConsoleA, 'WriteConsoleA', \

ReadConsoleA, 'ReadConsoleA', ExitProcess, 'ExitProcess'

section '.data' data readable writeable

memA Dd 200307.21 ;делимое

memB Dd 157202.5 ;делитель

memS Dd 0 ; переменная для числа записи числа S

dop Dd 0.5

conTitle db 'Console', 0

mes1 db 'Number A is equal to number S', 0dh, 0ah, 0

mes1Len = $-mes1

mes2 db 'Number A is less then number S', 0dh, 0ah, 0

mes2Len = $-mes2

mes3 db 'Number A is greater then number S', 0dh, 0ah, 0

mes3Len = $-mes3

hStdIn dd 0

hStdOut dd 0

chrsRead dd 0

chrsWritten dd 0

STD\_INP\_HNDL dd -10

STD\_OUTP\_HNDL dd -11

section '.text' code readable executable

Start:

invoke SetConsoleTitleA, conTitle

test eax, eax

jz Exit

invoke GetStdHandle, [STD\_OUTP\_HNDL]

mov [hStdOut], eax

invoke GetStdHandle, [STD\_INP\_HNDL]

mov [hStdIn], eax

fld [memB] ; записываем memB в вершину стека

fsin ; перезаписываем в ST0 значение sin(ST0)=sin(memB)

fld [memA] ; добавляем в стек memA, теперь ST0 = memA, ST1 = sin(memB)

fsub ST0, ST1 ; производим вычитание, ST0 = ST0 - ST1 = A - sin(B)

fadd [dop] ; добавляем к ST0 содержимое dop, теперь ST0 = S = A + 1/2 - sin(B)

fst [memS] ; записываем результат в переменную S

fld [memA] ; вновь добавляем в стек memA, теперь ST0 = A, ST1 = S

fcomi ST0, ST1 ; производим сравнение ST0 и ST1

jz Equal ; в случае ST0 = ST1 переходим на метку Equal

jc Less ; в случае ST0 < ST1 переходим на метку Less

invoke WriteConsoleA, [hStdOut], mes3, mes3Len, chrsWritten, 0

JMP Exit

Equal:

invoke WriteConsoleA, [hStdOut], mes1, mes1Len, chrsWritten, 0

JMP Exit

Less:

invoke WriteConsoleA, [hStdOut], mes2, mes2Len, chrsWritten, 0

JMP Exit

Exit:

invoke ReadConsoleA, [hStdIn], readBuf, 1, chrsRead, 0

invoke ExitProcess, 0

## Логика кода

В начале программы нам необходимо получить число S. Далее уже произвести сравнения.

Для этого мы записываем в вершину стека переменную memB, после чего легко получаем значение sin(memB) с помощью команды fsin, которая перезаписывает в ST(0) результат sin(ST(0)).

Далее мы добавляем в стек переменную memA, из-за чего значения в стеке съезжают вниз: в вершине теперь будет перезаписана ST(0) = memA, а предыдущее слагаемое будет перезаписано как ST(1) = sin(memB).

Далее нам остаётся только вычесть из числа memA значение sin(B), а затем прибавить константу 0,5. Для этого используем две команды:

* FSUB ST(0), ST(1)  
  Перезаписывает содержимое ST(0) на значение ST(0) – ST(1);
* FADD [dop]  
  Перезаписывает вершину стека как ST(0) = ST(0) + dop.

Теперь сохраняем полученное значение S в ОП в переменной memS с помощью команды fst [memS].

Под конец добавляем в вершину стека содержимое переменной memA для её сравнения с числом S, которое теперь будет находиться в ST(1).

Сравнение осуществляем с помощью команды FCOMI ST0, ST1. В зависимости от результата выводим соответствующее сообщение: больше, меньше или равно.

## Разбор инструкций

### Запись переменной в вершину стека

Перейдём к пользовательскому коду и начнём рассматривать инструкции работы со стеком. Первая инструкция касается записи переменной memB в вершину стека (рис. 4).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок . Запись memB в стек.

Из рисунка видно, что происходит обращение к ячейке памяти по адресу [403004], перейдём к ней и убедимся, что это действительно переменная memB (рис. 5).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок . Запись memB в ОП.

memB содержит число 157202.5, с помощью онлайн перевода рассмотрим, как бы записывалось это число в IEEE754 (рис. 6).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок . memB в формате IEEE754.

Остаётся только учесть, что в компьютере числа записываются с младших байтов. Значит, число memB в памяти компьютера будет записано как: , что полностью совпадает с результатом на рис. 5.

Рассмотрим результат выполнения данной инструкции. Так, до её выполнения стек выглядел согласно рисунку 7.

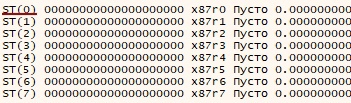


Рисунок . Стек до выполнения программы.

После выполнения инструкции в ST(0) записывается переменная memB (рис. 8).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок . Запись в стек memB.

Из рисунка можно заметить, что число расширилось до полного заполнения ячейки ST(0). Однако, поскольку дробная часть числа 0.5 переводится в 2СС без затруднений, никаких погрешностей мы здесь не наблюдаем.

### Использование FSIN

Далее в программе выполняется команда вычисления синуса fsin (рис. 9).



Рисунок . Вычисление синуса memB и перезапись ST(0).

После её выполнения содержимое ST(0) заменится на значение sin(ST0), что можно наблюдать на рисунке 10.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок . Выполнение fsin.

### Добавление новой переменной в стек

Следующей командой мы записываем в стек содержимое переменной memA (рис. 11).



Рисунок . Инструкция записи memA в стек.

После её выполнения содержимое стека изменяется (см. рис. 12). Можно заметить, что адрес прежней вершины стека не изменился и сдвинулся вниз, изменилось лишь его обозначение как ST(1). Более того, стоит обратить внимание на то, что в стеке всего 8 доступных ячеек, которые последовательно перемещаются при добавлении в стек новых значений.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок . Запись memA в ST(0).

Также обратим внимание на содержимое ST(0), в которой хранится число memA. В действительности число memA = 200307.21, однако в стек оно записалось со значительной погрешностью. Дело в том, что наше число при представлении в двоичной СС в формате ЧПЗ компьютер смог получить только приближённо. При этом из-за того, что размер числа (4 байта) меньше размера регистра, он расширился до 10 байт, поэтому мы и видим так много «лишних» цифр в ячейке стека.

Если посчитать, то погрешность составляет:

Перейдём к адресу памяти [403000], в котором хранится переменная memA (рис. 13). Значение «CD 9C 43 48» совпадает с правильно записанным числом memA = 48 43 9C CD (в компьютере числа хранятся с младшего байта).

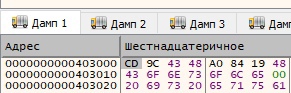


Рисунок . Значение memA.

### Команда вычитания FSUB

Следующая инструкция производит вычитание элементов стека и перезаписывает результат в ST(0) (рис. 14).



Рисунок . Инструкция вычитания.

В данной инструкции присутствует два явных операнда:

1. операнд-назначения, он же ST(0): из него будет производиться вычитание и на его место будет записан результат;
2. операнд-источник ST(1): он будет вычитаться из операнда-назначения.

Таким образом, после выполнения данной инструкции, в вершину стека будет записан результат операции ST(0) – ST(1), результат которой можно наблюдать на рисунке 15.



Рисунок . Стек после операции инструкции FSUB.

### Команда сложения FADD

Далее производится инструкция сложения (рис. 17). Данная команда уже по умолчанию работает только с вершиной стека и именно её перезаписывает. В нашем случае в качестве операнда мы указываем переменную dop, которая хранится в памяти по адресу [40300C] и которая после выполнения данной операции будет прибавлена к значению ST(0).



Рисунок . Инструкция FADD.

В переменной dop хранится число 0.5, поэтому если перейти к дампу памяти [40300C] (рис. 17), мы увидим значение “00 00 00 3F”, которое при перезаписи и переводе даёт нам: .

Сам результат выполнения инструкции можно наблюдать на рисунке 18.

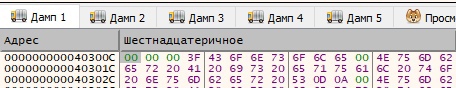


Рисунок . Содержимое переменной dop.

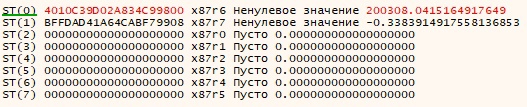


Рисунок . Результат выполнения FADD.

### Использование FST для записи ST

Следующая инструкция копирует содержимое вершины стека в переменную memS, которая хранится по адресу [403008] (рис. 19).



Рисунок . Инструкция записи FST.

Как видно на рисунке, у данной инструкции есть только один явный операнд – адрес памяти, куда будет записываться содержимое. Второй операнд является неявным и обозначает, откуда будет браться значение записи – вершина стека ST0.

После выполнения этой команды, в дампе памяти [403008] будет записано следующее (рис. 20):

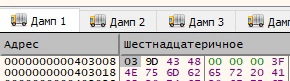


Рисунок . memS после записи ST.

С помощью онлайн перевода посмотрим, что это за число в представлении компьютера (рис. 21):

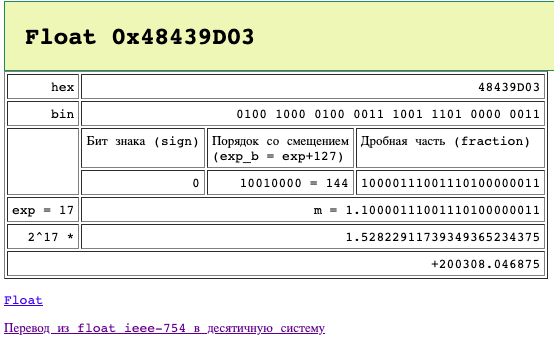


Рисунок . Перевод IEEE754 в 10 СС.

Для перевода мы записали содержимое memS, начиная со старших байтов (перевернули его), а в качестве результата получили число «200308.046875», что совпадает с маленькой погрешностью со значением ST(0) на рисунке Рисунок 18. Результат выполнения FADD..

После записи memS снова выполняется инструкция по добавлению в стек переменной memA, поэтому значение memS переедет вниз на ST(1), а в ST(0) будет записана memA (рис. 22).

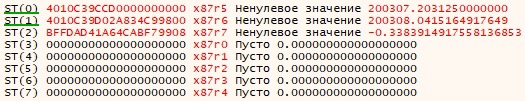


Рисунок . Добавление memA в вершину стека.

### Инструкция сравнения FCOMI

Наконец, производится сравнение двух верхних регистров стека ST0 и ST1, т.е. сравниваются числа A и S (рис. 23).



Рисунок . Сравнение FCOMI.

Посмотрим, как изменяются регистры флагов после выполнения этой команды. Так, на рисунке 24 изображены флаги до её выполнения, а на рис. 25 – после.



Рисунок . Регистры флагов до FCOMI.



Рисунок . Регистры флагов после FCOMI.

Как видно из рисунков, комбинация интересующих нас флагов составляет: ZF = 0, PF = 0, CF = 1. Это характерно для случая ST(0) < ST(1).

Значит, memA < memS, то есть A < S.

### Результат выполнения программы

В соответствии с полученными флагами наша программа перейдёт в секцию “Less” с помощью условного перехода “jc Less” и выведется соответствующее сообщение о результате (рис. 26).

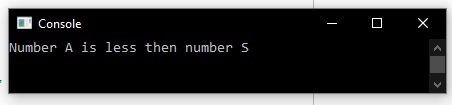


Рисунок . Результат выполнения программы Intel.

# Код MIPS

Теперь выполним ту же задачу в MIPS. Однако, поскольку процессор MIPS не поддерживает вычисление таких функций как sin(), программно автоматизируем этот процесс.

Так, для вычисления синуса можно воспользоваться рядом Тейора (см. рис. 27).

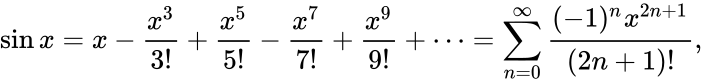


Рисунок . Разложение sin в числовой ряд.

Для получения синуса в коде воспользуемся циклом. На основе вышеописанного числового ряда можно использовать следующий алгоритм вычисления sin(x):

* S – общая сумма, на начальном этапе S = 0;
* n – показатель степени, n = 1;
* r – текущий член ряда, r = x;
* каждый последующий член ряда вычисляется как:
* при этом на каждой итерации S = S + r.

## Ссылка на github с кодом MIPS

Код программы на MIPS получимся чересчур объёмным, потому на случай его некорректного отображения оставляю ссылку на свой полный код, который я разместил в своём репозитории:

<https://github.com/marato-o/CS-CN/blob/master/lab_3/MIPS_lab_3.asm>

Сам код приведён ниже.

## Код

.data

memA: .float 200307.21

memB: .float 157202.5

memS: .float 0 # переменная для записи значения числа S

esp: .float 1.0e-15 # точность вычисления SIN

pi2: .float 6.283185307 # pi\*2

border: .float 10.0 # крайнее значение для memB, используется для условия вычисленя #тождественного memB

dop: .float 0.5 # понадобится при вычислении числа S

h1: .asciiz "\n"

mes\_sin: .asciiz "sin(B) = "

mes\_A: .asciiz "A = "

mes\_B: .asciiz "B = "

mes\_S: .asciiz "S = "

mes\_res: .asciiz "\nResult: "

mes\_equal: .asciiz "number A is equal to number S\n"

mes\_less: .asciiz "number A is less then number S\n"

mes\_greater: .asciiz "number A is greater then number S\n"

.text

#--------------------------- вычисление SIN(memB) -----------------------------#

SIN:

# для корректного вычисления синуса, при необходимости

# преобразуем большое число memB в меньшее ему тождественное

# используем правило sin(B) = sin(B - 2\*pi\*k), где k - целое число

lwc1 $f0, memB # помещаем memB в регистр f0

lwc1 $f1, border

c.lt.s $f1, $f0 # производим сравнение

bc1t FindEqual # при большом memB переходим к определению тождественного ему числа

Continue: # переход к началу вычисления sin(memB)

li $t0, 1 # счетчик n = 1

lwc1 $f1, esp

mov.s $f2, $f0 # здесь храним актуальный член ряда r, r\_0 = x = memB

la $a0, mes\_sin # помещаем в регистр сообщение mes\_sin

li $v0, 4

syscall # выводим на сообщение экран

Loop:

abs.s $f3, $f2 # берём модуль от члена ряда и записываем его в f3

c.lt.s $f3, $f1 # сверяем член ряда с минимальной точностью

bc1t LabTask # если член ряда по-прежнему больше нужной точности, цикл продолжится

add.s $f12, $f12, $f2 # добавляем к общей сумме член ряда

add $t0, $t0, 2 # увеличиваем степень n на 2

sub $t1, $t0, 1 # вычитаем из степени 1 и результат сохраняем в $t1

mul $t1, $t1, $t0 # рассчитываем произведение n\*(n-1)

mtc1 $t1, $f4 # помещаем содержимое из $t1 в $f4

cvt.s.w $f4, $f4 # переводим целое число n\*(n-1) в ЧПЗ

div.s $f4, $f0, $f4 # делим x/(n\*(n-1))

mul.s $f4, $f4, $f0 # умножаем результат на x

neg.s $f4, $f4 # домножаем результат на (-1)

mul.s $f2, $f2, $f4 # умножаем полученное на предыдущий член ряда

# в результате получаем, что f2 = -r\*x\*x/(n\*(n-1))

j Loop # возвращаемся к началу цикла

FindEqual:

lwc1 $f1, pi2 # записываем в регистр константу pi\*2

div.s $f2, $f0, $f1 # делим memB/(pi\*2) и запоминаем результат в f2

# теперь нужно округлить полученное число вниз

# для этого проще конвертировать float в int, отбросив дробную часть

cvt.w.s $f2, $f2 # преобразуем f2 в целое число и перезаписываем его в f2

cvt.s.w $f2, $f2 # переводим целое число обратно в ЧПЗ

mul.s $f1, $f1, $f2 # перезаписываем в f1 произведение от (2\*pi) на их #рассчитанное целое кол-во

sub.s $f0, $f0, $f1 # вычитаем из memB полученное число и записываем результат. #Это и есть наименьшее тождественное число

j Continue

#-------------------------- начало выполнения задачи ЛР3 -------------------------#

LabTask:

li, $v0, 2

syscall # выводим на экран содержимое регистра f12 = sin(B)

la $a0, h1

li, $v0, 4

syscall # производим перенос строки (выводим сообщение "\n")

# теперь рассчитываем S = A + 0.5 - sin(B)

# memA будем хранить в регистре f0

# dop = 0.5 будем хранить в f1

# значение sin(B) будем по-прежнему использовать из регистра f12

lwc1 $f0, memA

lwc1 $f1, dop

add.s $f2, $f0, $f1

sub.s $f2, $f2, $f12

swc1 $f2, memS # записываем результат в переменную memS

#------------------------- выводим значения чисел A, B и S -----------------------#

la $a0, mes\_A

li, $v0, 4

syscall # выводим на экран сообщение mes\_A

mov.s $f12, $f0 # записываем в f12 значение числа A

li, $v0, 2

syscall # выводим на экран содержимое регистра f12 = sin(B)

la $a0, h1

li, $v0, 4

syscall # производим перенос строки (выводим сообщение "\n")

la $a0, mes\_B

li, $v0, 4

syscall # выводим на экран сообщение mes\_B

lwc1 $f12, memB # записываем в f12 значение числа B

li, $v0, 2

syscall # выводим на экран содержимое регистра f12 = B

la $a0, h1

li, $v0, 4

syscall # производим перенос строки (выводим сообщение "\n")

la $a0, mes\_S

li, $v0, 4

syscall # выводим на экран сообщение mes\_S

lwc1 $f12, memS # записываем в f12 значение числа S

li, $v0, 2

syscall # выводим на экран содержимое регистра f12 = S

la $a0, h1

li, $v0, 4

syscall # производим перенос строки (выводим сообщение "\n")

#-------------------- здесь производим сравнения чисел A и S ----------------#

la $a0, mes\_res

li, $v0, 4

syscall

# memA хранится в $f0

# memS по-прежнему находится в $f12 и в $f2

c.eq.s $f0, $f12

bc1t Equal

c.lt.s $f0, $f12

bc1t Less

Greater:

la $a0, mes\_greater

li, $v0, 4

syscall

j Exit

Equal:

la $a0, mes\_equal

li, $v0, 4

syscall

j Exit

Less:

la $a0, mes\_less

li, $v0, 4

syscall

Exit:

li, $v0, 10

syscall # завершаем программу

## Результат выполнения программы

После выполнения программного кода выведутся следующие сообщения (см. рис. 28).

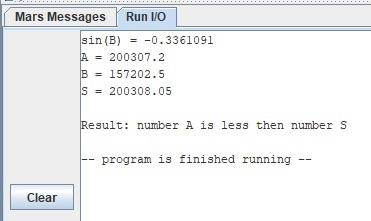


Рисунок . Результат выполнения программы MIPS.

## Разбор инструкций

При работе в MIPS с вещественными числами необходимо использовать регистры $f0 - $f32, каждый из которых предназначен для хранения чисел типа float.

Так, для выполнения задания, мы сохраняем содержимое переменных в регистрах, после чего производим арифметические операции и сравнения.

### Разбор основных инструкций

Разберём в отладчике начало программы и часть цикла вычисления функции SIN.

В самом начале мы добавляем содержимое memB в регистр $f0 (рис. 29).

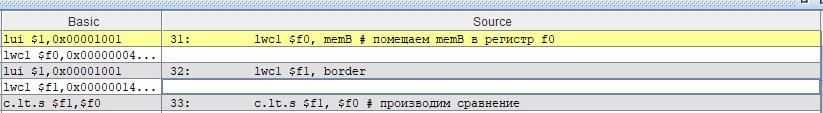


Рисунок . Инструкция добавления memB в $f0.

После выполнения этой инструкции в регистре можно обнаружить следующее: (см. рис. 30).

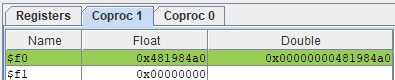


Рисунок . Регистр $f0 после записи memB.

Данное число уже разбиралось при работе с Intel (см. рис. Рисунок 6. memB в формате IEEE754.).

Далее происходит запись переменной border в регистр $f1, результат которой можно наблюдать на рис. 31.

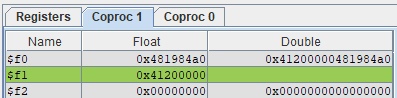


Рисунок . Регистр $f1 после записи border.

Следующей важной инструкцией идёт сравнение этих двух регистров (рис. 32). В результате сравнения выяснится, достаточно ли большим является число memB для запуска процедуры вычисления меньшего тождественного ему числа.

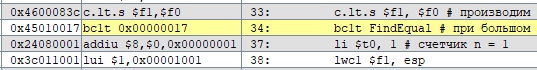


Рисунок . Сравнение $f1 и $f0.

После выполнения данной инструкции позиция флага «0» (по умолчанию именно с этим флагом проходит работа) изменится, если условие   
$f1 <  $f0 выполняется. Как видно из рисунка 33, позиция флага изменилась, а потому произойдёт переход к метке FindEqual.

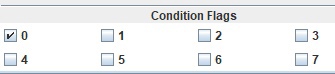


Рисунок . Изменение флага операции сравнения.

Разберём инструкции после этого перехода. Каждую команду можно наблюдать на рисунке 34.

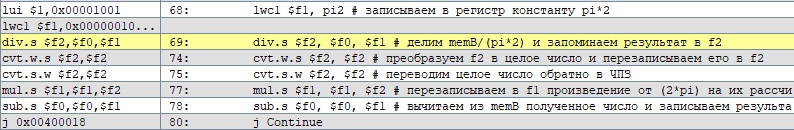


Рисунок . Инструкции секции FindEqual.

В начале в регистр f1 перезапишется переменная pi2, а затем будет выполнено деление memB/f1 в инструкции 69, результат деления запишется в регистр f2 (рис. 35). Далее идут наиболее важные две инструкции.



Рисунок . $f2 после операции деления.

Для того, чтобы получить минимальное тождественное число из равенства sin(B) = sin(B – 2\*pi\*k), необходимо определить максимальное целое число k. Для этого осталось только отбросить дробную часть от f2, поэтому следующие две инструкции сначала переводят содержимое f2 из float в word (ЧПЗ в int), отбрасывая дробную часть, а после снова переводят это число в float.

Результат выполнения инструкции 74 можно наблюдать на рисунке 36, а результат от выполнения 75ой инструкции – на рисунке 37.



Рисунок . $f2 после конвертирования в целое число.



Рисунок . $f2 после обратной конвертации в float.

При сравнении рисунков 35 и 37 хорошо видно, что старший байт числа обнулился и число немного изменилось.

Последние две инструкции в этом блоке перезаписывают регистр $f0 на минимальное тождественное значение memB (рис. 38). Далее происходит прыжок на метку Continue, которая содержит вычисление функции sin(B).



Рисунок . Инструкции перезаписи $f0 на новое memB.

Как видно из рисунка, прыжок произойдёт на адрес инструкции [0x00400018], после выполнения этой команды (рис. 39) мы перейдём на инструкцию, адрес которой совпадает с названным ранее.

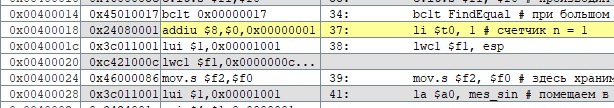


Рисунок . Результат перехода на блок с адресом [0x00400018].

Далее происходит расчёт значения синуса по названному ранее алгоритму. В связи с достаточно большим количеством инструкций, логику которых мы уже разобрали, анализ кода на этом закончим.