

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
Перевод числа в формат IEEE754-2008.....	3
Решение	3
Проверка результатов	5
Условие задачи	5
Код Intel	6
Ссылка на github с кодом Intel	6
Код.....	6
Логика кода	7
Разбор инструкций.....	8
Запись переменной в вершину стека.....	8
Использование FSIN	10
Добавление новой переменной в стек.....	10
Команда вычитания FSUB	11
Команда сложения FADD	11
Использование FST для записи ST	12
Инструкция сравнения FCOMI.....	14
Результат выполнения программы	14
Код MIPS	15
Ссылка на github с кодом MIPS	15
Код.....	15
Результат выполнения программы.....	19
Разбор инструкций.....	19
Разбор основных инструкций	19

Условие для перевода

Дата моего дня рождения: 21.07.2003

Поскольку $a_6 = 7$ является нечётным, число A будет положительным.

Число $A = 200307.21_{10}$

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:

$$200307_{10} = 110000111001110011_2$$

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

Введите число

Его система счисления	Перевести в
Двоичная <input type="radio"/>	<input checked="" type="radio"/> Двоичную
Троичная <input type="radio"/>	<input type="radio"/> Троичную
Восьмеричная <input type="radio"/>	<input type="radio"/> Восьмеричную
Десятичная <input checked="" type="radio"/>	<input type="radio"/> Десятичную
Шестнадцатеричная <input type="radio"/>	<input type="radio"/> Шестнадцатеричную
Двоично-десятичная <input type="radio"/>	<input type="radio"/> Двоично-десятичную
Другая <input type="radio"/>	<input type="radio"/> Другую

Результат:

110000111001110011

Рисунок 1. Подсчёт целой части числа A .

Дробная часть:

$$0.21_{10} = 0.00110101110_2$$

Таким образом:

$$A = 110000111001110011.00110101110_2$$

Введите число

Его система счисления	Перевести в
Двоичная <input type="radio"/>	<input checked="" type="radio"/> Двоичную
Троичная <input type="radio"/>	<input type="radio"/> Троичную
Восьмеричная <input type="radio"/>	<input type="radio"/> Восьмеричную
Десятичная <input checked="" type="radio"/>	<input type="radio"/> Десятичную
Шестнадцатеричная <input type="radio"/>	<input type="radio"/> Шестнадцатеричную
Двоично-десятичная <input type="radio"/>	<input type="radio"/> Двоично-десятичную
Другая <input type="radio"/>	<input type="radio"/> Другую

ПЕРЕВЕСТИ

Результат:

0.00110101110

Рисунок 2. Подсчёт дробной части числа А.

2) Запишем число в нормализованной виде:

$$1.1000011100111001100110101110_2 \cdot 2^{17}$$

3) Определим мантиссу:

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

$$\underbrace{1.1000011100111001100110101110_2}_{25 \text{ разрядов числа}} + 0$$

$$\underbrace{1.10000111001110011001101}_{23 \text{ разряда мантиссы без старшей "1"}}$$

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

$$m = 10000111001110011001101_2$$

4) Определим порядок:

$$p = 17 + 127 = 144 = 10010000_2$$

5) Результат

$$\underbrace{0.10010000}_{4} \underbrace{.10000111001110011001101}_{8 \quad 4 \quad 3 \quad 9 \quad C \quad C \quad D}$$

Таким образом: $A = 48\ 43\ 9C\ CD_{16}$

Проверка результатов

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

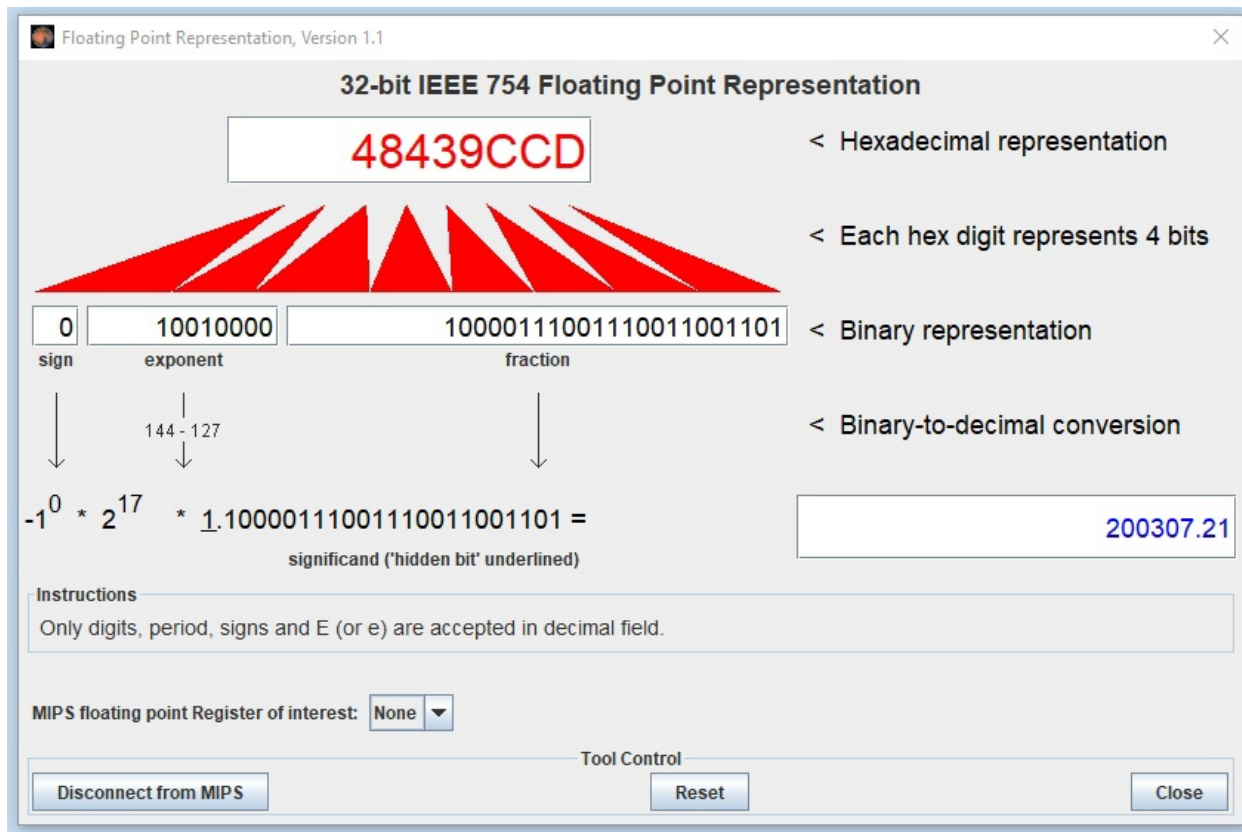


Рисунок 3. Проверка результатов перевода в MIPS.

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 $S = A + 1/2 - \sin(B)$.
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. После чего уже произвести сравнения.

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

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

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 $\text{ST}(0) - \text{ST}(1)$;
- FADD [dop]
Перезаписывает вершину стека как $\text{ST}(0) = \text{ST}(0) + \text{dop}$.

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).

<code>fld [memB]</code>		
0000000000404051	D905 ADEFFFFFFF	<code>fld st(0), dword ptr ds:[403004]</code>
0000000000404057	D9FE	<code>fsin</code>
0000000000404059	D905 A1EFFFFFFF	<code>fld st(0), dword ptr ds:[403000]</code>
000000000040405F	D8E1	<code>fsub st(0), st(1)</code>
0000000000404061	D805 A5EFFFFFFF	<code>fadd st(0), dword ptr ds:[40300C]</code>
0000000000404067	D915 9BEFFFFFFF	<code>fst dword ptr ds:[403008], st(0)</code>
000000000040406D	D905 8DEFFFFFFF	<code>fld st(0), dword ptr ds:[403000]</code>
0000000000404073	DBF1	<code>fcomi st(0), st(1)</code>
0000000000404075	74 36	<code>je lab_2.4040AD</code>
0000000000404077	72 68	<code>jb lab_2.4040E1</code>
0000000000404079	48:83EC 30	<code>sub rsp, 30</code>

Рисунок 4. 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).




 Дамп 1	 Дамп 2	 Дамп 3
Адрес		Шестнадцатеричное
0000000000403004		A0 84 19 48 00 00
0000000000403014		6F 6C 65 00 4E 75

Рисунок 5. Запись memB в ОП.

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

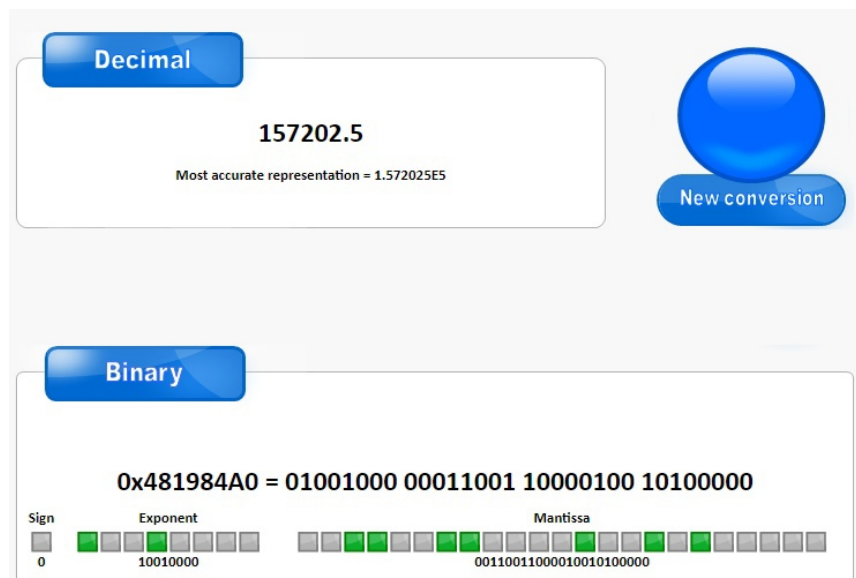


Рисунок 6. memB в формате IEEE754.

Остаётся только учесть, что в компьютере числа записываются с младших байтов. Значит, число memB в памяти компьютера будет записано как: A0 84 19 48, что полностью совпадает с результатом на рис. 5.

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

<u>ST(0)</u>	00000000000000000000	x87r0	Пусто	0.000000000
ST(1)	00000000000000000000	x87r1	Пусто	0.000000000
ST(2)	00000000000000000000	x87r2	Пусто	0.000000000
ST(3)	00000000000000000000	x87r3	Пусто	0.000000000
ST(4)	00000000000000000000	x87r4	Пусто	0.000000000
ST(5)	00000000000000000000	x87r5	Пусто	0.000000000
ST(6)	00000000000000000000	x87r6	Пусто	0.000000000
ST(7)	00000000000000000000	x87r7	Пусто	0.000000000

Рисунок 7. Стек до выполнения программы.

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

<u>ST(0)</u>	40109984A00000000000	x87r7	Ненулевое значение	157202.50000000000000
ST(1)	00000000000000000000	x87r0	Пусто	0.00000000000000000000
ST(2)	00000000000000000000	x87r1	Пусто	0.00000000000000000000
ST(3)	00000000000000000000	x87r2	Пусто	0.00000000000000000000
ST(4)	00000000000000000000	x87r3	Пусто	0.00000000000000000000
ST(5)	00000000000000000000	x87r4	Пусто	0.00000000000000000000
ST(6)	00000000000000000000	x87r5	Пусто	0.00000000000000000000
ST(7)	00000000000000000000	x87r6	Пусто	0.00000000000000000000

Рисунок 8. 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).

0000000000404057	D9FE	fsin
0000000000404059	D905 A1EFFFFF	fld st(0),dword ptr ds:[403000]
000000000040405F	D8E1	fsub st(0),st(1)

Рисунок 9. Вычисление синуса memB и перезапись ST(0).

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

ST(0)	BFFDAD41A64CABF79908	x87r7	Ненулевое значение	-0.3383914917558136853
ST(1)	00000000000000000000	x87r0	Пусто	0.00000000000000000000
ST(2)	00000000000000000000	x87r1	Пусто	0.00000000000000000000
ST(3)	00000000000000000000	x87r2	Пусто	0.00000000000000000000
ST(4)	00000000000000000000	x87r3	Пусто	0.00000000000000000000
ST(5)	00000000000000000000	x87r4	Пусто	0.00000000000000000000
ST(6)	00000000000000000000	x87r5	Пусто	0.00000000000000000000
ST(7)	00000000000000000000	x87r6	Пусто	0.00000000000000000000

Рисунок 10. Выполнение fsin.

Добавление новой переменной в стек

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

0000000000404059	D905 A1EFFFFF	fld st(0),dword ptr ds:[403000]
000000000040405F	D8E1	fsub st(0),st(1)
0000000000404061	D805 A5EFFFFF	fadd st(0),dword ptr ds:[40300C]

Рисунок 11. 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.

ST(0)	4010C39CCD0000000000	x87r6	Ненулевое значение	200307.2031250000000
ST(1)	BFFDAD41A64CABF79908	x87r7	Ненулевое значение	-0.3383914917558136853
ST(2)	00000000000000000000	x87r0	Пусто	0.00000000000000000000
ST(3)	00000000000000000000	x87r1	Пусто	0.00000000000000000000
ST(4)	00000000000000000000	x87r2	Пусто	0.00000000000000000000
ST(5)	00000000000000000000	x87r3	Пусто	0.00000000000000000000
ST(6)	00000000000000000000	x87r4	Пусто	0.00000000000000000000
ST(7)	00000000000000000000	x87r5	Пусто	0.00000000000000000000

Рисунок 12. 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: $0.21 - 0.203125 = 0.0060875$

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).





 Дамп 1	 Дамп 2	 Дамп 3	 Дамп 4						
Адрес		Шестнадцатеричное							
0000000000403000		CD	9C	43	48	A0	84	19	48
0000000000403010		43	6F	6E	73	6F	6C	65	00
0000000000403020		20	69	73	20	65	71	75	61

Рисунок 13. Значение memA.

Команда вычитания FSUB

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

000000000040405F	D8E1	fsub st(0),st(1)
0000000000404061	D805 A5EFFFFF	fadd st(0),dword ptr ds:[40300C]

Рисунок 14. 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.

ST(0)	4010C39CE2A834C99800	x87r6	Ненулевое значение	200307.5415164917649
ST(1)	BFFDAD41A64CABF79908	x87r7	Ненулевое значение	-0.3383914917558136853
ST(2)	00000000000000000000	x87r0	Пусто	0.00000000000000000000
ST(3)	00000000000000000000	x87r1	Пусто	0.00000000000000000000
ST(4)	00000000000000000000	x87r2	Пусто	0.00000000000000000000
ST(5)	00000000000000000000	x87r3	Пусто	0.00000000000000000000
ST(6)	00000000000000000000	x87r4	Пусто	0.00000000000000000000
ST(7)	00000000000000000000	x87r5	Пусто	0.00000000000000000000

Рисунок 15. 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 ptr, которая хранится в памяти по адресу [40300C] и которая после выполнения данной операции будет прибавлена к значению ST(0).

0000000000404061	D805 A5EFFFFF	fadd st(0), dword ptr ds:[40300C]
0000000000404067	D915 9BEFFFFF	fst dword ptr ds:[403008], st(0)
000000000040406D	D905 8DEFFFFF	fld st(0), dword ptr ds:[403000]

Рисунок 16. Инструкция FADD.

В переменной dor 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: $3F000000_{16} = 0.5_{10}$.

Сам результат выполнения инструкции можно наблюдать на рисунке 18.







 Дамп 1	 Дамп 2	 Дамп 3	 Дамп 4	 Дамп 5	 Просм											
Адрес	Шестнадцатеричное															
000000000040300C	00	00	00	3F	43	6F	6E	73	6F	6C	65	00	4E	75	6D	62
000000000040301C	65	72	20	41	20	69	73	20	65	71	75	61	6C	20	74	6F
000000000040302C	20	6E	75	6D	62	65	72	20	53	0D	0A	00	4E	75	6D	62

Рисунок 17. Содержимое переменной dor.

ST(0)	4010C39D02A834C99800	x87r6	Ненулевое значение	200308.0415164917649
ST(1)	BFFDAD41A64CABF79908	x87r7	Ненулевое значение	-0.3383914917558136853
ST(2)	00000000000000000000	x87r0	Пусто	0.00000000000000000000
ST(3)	00000000000000000000	x87r1	Пусто	0.00000000000000000000
ST(4)	00000000000000000000	x87r2	Пусто	0.00000000000000000000
ST(5)	00000000000000000000	x87r3	Пусто	0.00000000000000000000
ST(6)	00000000000000000000	x87r4	Пусто	0.00000000000000000000
ST(7)	00000000000000000000	x87r5	Пусто	0.00000000000000000000

Рисунок 18. Результат выполнения FADD.

Использование FST для записи ST

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

0000000000404067	D915 9BEFFFFF	fst dword ptr ds:[403008], st(0)
000000000040406D	D905 8DEFFFFF	fld st(0), dword ptr ds:[403000]

Рисунок 19. Инструкция записи FST.

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

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

Дамп 1	Дамп 2	Дамп 3	Дамп 4
Адрес	Шестнадцатеричное		
00000000000403008	03	9D	43 48 00 00 00 3F
00000000000403018	4E	75	6D 62 65 72 20 41

Рисунок 20. memS после записи ST.

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

Float 0x48439D03			
hex	48439D03		
bin	0100 1000 0100 0011 1001 1101 0000 0011		
	Бит знака (sign)	Порядок со смещением (exp_b = exp+127)	Дробная часть (fraction)
	0	10010000 = 144	10000111001110100000011
exp = 17	m = 1.10000111001110100000011		
2^17 *	1.52822911739349365234375		
+200308.046875			

[Float](#)

[Перевод из float ieee-754 в десятичную систему](#)

Рисунок 21. 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 18.

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).

```

ST(0) 4010C39CCD0000000000 x87r5 Ненулевое значение 200307.2031250000000
ST(1) 4010C39D02A834C99800 x87r6 Ненулевое значение 200308.0415164917649
ST(2) BFFDAD41A64CABF79908 x87r7 Ненулевое значение -0.3383914917558136853
ST(3) 00000000000000000000 x87r0 Пусто 0.00000000000000000000
ST(4) 00000000000000000000 x87r1 Пусто 0.00000000000000000000
ST(5) 00000000000000000000 x87r2 Пусто 0.00000000000000000000
ST(6) 00000000000000000000 x87r3 Пусто 0.00000000000000000000
ST(7) 00000000000000000000 x87r4 Пусто 0.00000000000000000000

```

Рисунок 22. 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).

0000000000404073	DBF1	fcomi st(0),st(1)
0000000000404075	74 36	je lab_2.4040AD
0000000000404077	72 68	jb lab_2.4040E1

Рисунок 23. 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.

```
RFLAGS 0000000000000206
ZF 0 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1
```

Рисунок 24. Регистры флагов до FCOMI.

```
RFLAGS 0000000000000203
ZF 0 PF 0 AF 0
OF 0 SF 0 DF 0
CF 1 TF 0 IF 1
```

Рисунок 25. 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).

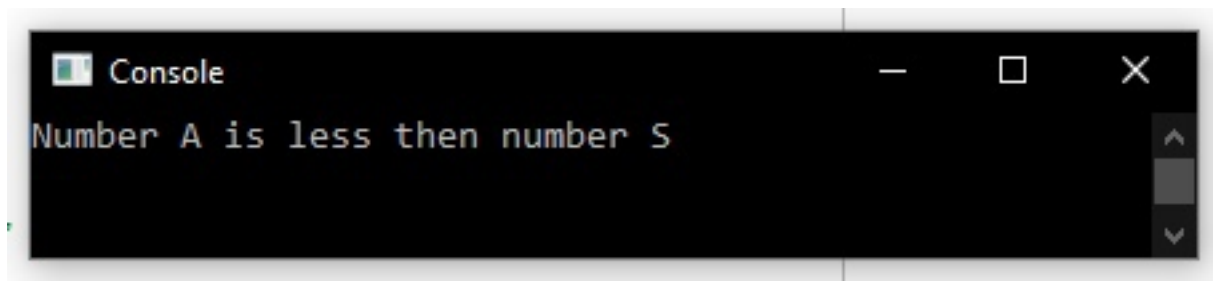


Рисунок 26. Результат выполнения программы Intel.

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 Тейлора (см. рис. 27).

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + \dots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!},$$

Рисунок 27. 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:

$$r_i = -r_{i-1} \cdot x \cdot x \cdot \frac{1}{n \cdot (n-1)}$$

- при этом на каждой итерации $S = S + r$.

Ссылка на github с кодом MIPS

Код программы на MIPS получился 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 вычисления
#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 ЛРЗ -----#
```

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).

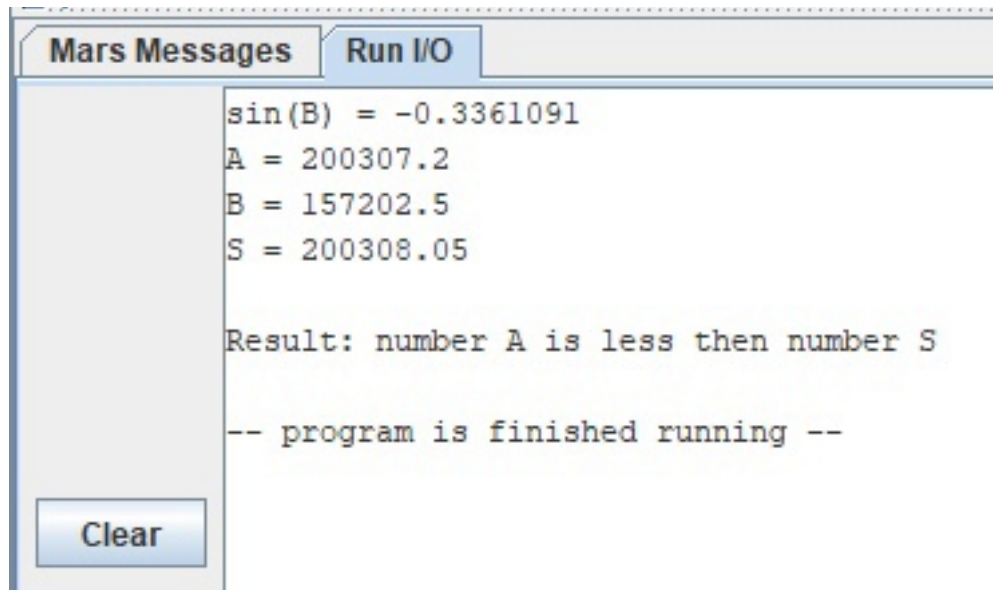


Рисунок 28. Результат выполнения программы MIPS.

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).

Basic	Source
lui \$1,0x00001001	31: lwcl \$f0, memB # помещаем memB в регистр f0
lwcl \$f0,0x00000004...	
lui \$1,0x00001001	32: lwcl \$f1, border
lwcl \$f1,0x00000014...	
c.lt.s \$f1,\$f0	33: c.lt.s \$f1, \$f0 # производим сравнение

Рисунок 29. Инструкция добавления memB в \$f0.

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

Registers	Coproc 1	Coproc 0
Name	Float	Double
\$f0	0x481984a0	0x00000000481984a0
\$f1	0x00000000	

Рисунок 30. Регистр \$f0 после записи memB.

Данное число уже разбиралось при работе с Intel (см. рис. 6).

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

Registers	Coproc 1	Coproc 0
Name	Float	Double
\$f0	0x481984a0	0x41200000481984a0
\$f1	0x41200000	
\$f2	0x00000000	0x0000000000000000

Рисунок 31. 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.

0x4600083c	c.lt.s \$f1,\$f0	33:	c.lt.s \$f1, \$f0 # производим
0x45010017	bclt 0x00000017	34:	bclt FindEqual # при большом
0x24080001	addiu \$8,\$0,0x00000001	37:	li \$t0, 1 # счетчик n = 1
0x3c011001	lui \$1,0x00001001	38:	lwcl \$f1, esp

Рисунок 32. 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.

Condition Flags			
<input checked="" type="checkbox"/> 0	<input type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3
<input type="checkbox"/> 4	<input type="checkbox"/> 5	<input type="checkbox"/> 6	<input type="checkbox"/> 7

Рисунок 33. Изменение флага операции сравнения.

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

lui \$l,0x00001001	68:	lwcl \$f1, pi2 # записываем в регистр константу pi*2
lwcl \$f1,0x00000010...		
div.s \$f2,\$f0,\$f1	69:	div.s \$f2, \$f0, \$f1 # делим memB/(pi*2) и запоминаем результат в f2
cvt.w.s \$f2,\$f2	74:	cvt.w.s \$f2, \$f2 # преобразуем f2 в целое число и перезаписываем его в f2
cvt.s.w \$f2,\$f2	75:	cvt.s.w \$f2, \$f2 # переводим целое число обратно в ЧПЗ
mul.s \$f1,\$f1,\$f2	77:	mul.s \$f1, \$f1, \$f2 # перезаписываем в f1 произведение от (2*pi) на их расщ
sub.s \$f0,\$f0,\$f1	78:	sub.s \$f0, \$f0, \$f1 # вычитаем из memB полученное число и записываем результа
j 0x00400018	80:	j Continue

Рисунок 34. 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.

\$f2	0x46c3771c	0x00000000046c3771c
------	------------	---------------------

Рисунок 35. \$f2 после операции деления.

Для того, чтобы получить минимальное тождественное число из равенства $\sin(B) = \sin(B - 2 \cdot \pi \cdot k)$, 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.

\$f1	0x40c90fdb	
\$f2	0x000061bb	0x0000000000000061bb
\$f3	0x00000000	

Рисунок 36. \$f2 после конвертирования в целое число.

\$f2	0x46c37600	0x00000000046c37600
------	------------	---------------------

Рисунок 37. \$f2 после обратной конвертации в float.

При сравнении рисунков 35 и 37 хорошо видно, что старший байт числа обнулится 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 $\sin(B)$.

mul.s \$f1,\$f1,\$f2	77:	mul.s \$f1, \$f1, \$f2 #
sub.s \$f0,\$f0,\$f1	78:	sub.s \$f0, \$f0, \$f1 #
j 0x00400018	80:	j Continue

Рисунок 38. 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.

0x00400014	0x45010017	bclt 0x00000017	34:	bclt FindEqual # при большом
0x00400018	0x24080001	addiu \$8,\$0,0x00000001	37:	li \$t0, 1 # счетчик n = 1
0x0040001c	0x3c011001	lui \$1,0x00001001	38:	lwcl \$f1, esp
0x00400020	0xc421000c	lwcl \$f1,0x0000000c...		
0x00400024	0x46000086	mov.s \$f2,\$f0	39:	mov.s \$f2, \$f0 # здесь храним
0x00400028	0x3c011001	lui \$1,0x00001001	41:	la \$a0, mes_sin # помещаем в

Рисунок 39. 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.