Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Архитектура вычислительных систем»

**ОТЧЕТ**

к лабораторной работе №1

на тему:

# «ПРОГРАММИРОВАНИЕ АРИФМЕТИЧЕСКОГО СОПРОЦЕССОРА»

БГУИР 1-40-04-01

|  |
| --- |
| Выполнил студент группы 253504  Дмитрук Богдан Ярославович |
|  |
| (дата, подпись студента) |
| Проверил ассистент кафедры информатики  Калиновская Анастасия Александровна |
|  |
| (дата, подпись преподавателя) |

Минск 2024

**Теоретические сведения:**

***Сопроцессорные конфигурации***

 Использование сопроцессора  позволяет значительно ускорить работу программ, выполняющих расчеты с высокой точностью, тригонометрические вычисления и обработку информации, которая должна быть представлена в виде действительных чисел. Сопроцессор подключается к системной шине параллельно с центральным процессором (CPU) и может работать только совместно с ним. Все команды попадают в оба процессора, а выполняет каждый свои. Сопроцессор не имеет своей программы и не может осуществлять выборку команд и данных. Это делает центральный процессор. Сопроцессор перехватывает с шины данные и после этого реализует конкретные действия по выполнению команды. Два процессора работают параллельно, что повышает эффективность системы.  Но возникают ситуации, когда их работа требует синхронизации (из-за разницы во времени выполнения команд).

**Синхронизация по командам**. Когда  центральный процессор выбирает для выполнения команду FPU, последний может быть занят выполнением предыдущей команды. Поэтому перед каждой командой сопроцессора в программе должна стоять специальная команда (wait), которая только проверяет текущее состояние FPU и, если он занят, переводит центральный процессор в состояние ожидания. Соответствующую команду в программу может вводить либо ассемблер, либо компилятор языка без указаний программиста.

**Синхронизация по данным**. Если выполняемая в  FPU команда записывает операнд в память перед последующей командой СРU, которая обращается к этой ячейке памяти, требуется команда проверки состояния FPU. Если данные еще не были записаны, СPU должен переходить в состояние ожидания. Автоматически учесть такие ситуации довольно сложно, поэтому вводить команды, которые проверяют состояние сопроцессора и при необходимости заставляют центральный процессор ожидать, должен программист.

***Программная модель сопроцессора***

В программную модель  любого процессора включаются только те регистры, которые доступны программисту на уровне машинных команд. Основу программной модели FPU образует регистровый стек из восьми 80-битных регистров R0-R7. В них хранятся числа в вещественном формате. В любой момент времени 3-битное поле ST  в слове состояния определяет регистр, являющийся текущей вершиной стека и обозначаемый ST(0). При занесении в стек (push) осуществляется декремент поля  ST и загружаются данные в новую вершину стека. При извлечении из стека (pop) в получатель, которым чаще всего является память, передается содержимое вершины стека, а затем инкрементируется поле  ST .

В организации регистрового стека FPU есть отличия от классического стека.

1.  Стек имеет кольцевую структуру. Контроль за использованием стека должен осуществлять программист. Максимальное число занесений без промежуточных извлечений равно 8.

2.    В командах FPU допускается явное или неявное обращение к регистрам с модификацией или без поля  ST. Например, команда fsqrt  замещает число из вершины стека значением корня из него. В бинарных операциях допускается явное указание регистров. Адресация осуществляется относительно текущей вершины стека и обозначение ST (i)  0<i<7 , считая от вершины.

   3. Не все стековые команды автоматически модифицируют указатель вершины стека.

С каждым регистром стека ассоциируется 2-битный тег (признак), совокупность которых образует слово тегов. Тег регистра R0 находиться в младших битах, R7 – в старших. Тег фиксирует наличие в регистре действительного числа (код 00), истинного нуля (код 01), ненормализованного или бесконечности (код 10) и отсутствие данных (код 11).  Наличие тегов позволяет  FPU быстрее обнаруживать особые случаи (попытка загрузить в непустой регистр, попытка извлечь из пустого) и обрабатывать данные.

Остальными регистрами FPU являются регистр управления, регистр состояния, два регистра состояния команды и два регистра указателя данных. Длина их всех 16 бит.

***Форматы численных данных***

 Арифметический FPU К1810ВМ87 оперирует с семью форматами численных данных, образующих три класса: двоичные целые, упакованные десятичные целые и вещественные числа. Во всех форматах старший (левый) бит отведен для знака.

Форматы различаются  длиной, следовательно, диапазоном допустимых чисел, способом представления (упакованный и неупакованный формат), способом кодировки(прямой и дополнительный код).

**Двоичные целые числа.**Три формата целых двоичных (целое слово (16 бит), короткое целое (32 бита), длинное целое (64 бита)) отличаются длиной, следовательно, диапазоном чисел. Только в этих форматах применяется стандартный дополнительный код. 0 имеет единственное кодирование. Наибольшее положительное число кодируеться как 011…1, а  Наибольшее по модулю отрицательное как 100..0.

**Упакованные десятичные целые.** Числа представлены в прямом коде и упакованном формате, т.е. в байте содержится две десятичные цифры в коде 8421. Старший бит левого байта – знак, остальные игнорируются, но при записи в них помещаются нули.  Но надо учитывать, что при наличии в тетраде запрещающих комбинаций 1010 – 1111 результат операции не определен. Т.е. сопроцессор не контролирует правильность результата.

**Вещественные числа.**Различают короткие вещественные (КВ)(мантиса – 24 бита, порядок – 8 бит), длинные вещественные (ДВ) (мантиса – 53 бита, порядок – 11 бит) и временные вещественные (ВВ) (мантиса – 64 бита, порядок – 15 бит). Для них применяется формат с плавающей точкой. Значащие цифры находятся в поле мантисы, порядок показывает фактическое положение двоичной точки в разрядах мантисы, бит знака S определяет знак числа.

**Цель работы:** Вариант 7. Значение аргумента **x** изменяется от ***a*** до ***b*** с шагом ***h***. Для каждого ***x*** найти значения функции Y(x), суммы S(x) и число итераций n, при котором достигается требуемая точность ε = |Y(x)-S(x)|. Результат вывести в виде таблицы. Значения ***a****,****b****,****h*** и ***ε*** вводятся с клавиатуры. E:\Uni\ACS\documents\Labs_7_8.files\image018.gif

**Ход работы:** на рисунке 1 представлен график зависимости времени выполнения от количества членов ряда с включенной функцией SMT, на рисунке 2 представлен график зависимости времени выполнения от количества членов ряда с выключенной функцией SMT.

Листинг 1 – Метод, вычисляющий сумму ряда

template<typename T>

T Calculator<T>::calculateS(T x, int n)

{

T result = 0;

int32\_t n2\_buffer = 0;

double result\_buffer = 0, decr = 1;

for (int i = 1; i <= n; ++i)

{

\_\_asm {

pushad;

finit

mov eax, i

imul eax, 2

dec eax

mov ecx, eax

mov n2\_buffer, eax

fld qword ptr[x]

fld st(0)

dec ecx

test ecx, ecx

je powerend

}

powerloop:

\_\_asm {

fmul st, st(1)

loop powerloop

}

powerend:

\_\_asm {

fild dword ptr [n2\_buffer]

sub dword ptr[n2\_buffer], 1

fild dword ptr[n2\_buffer]

mov ecx, dword ptr[n2\_buffer]

test ecx, ecx

je endfuc

}

faclup:

\_\_asm {

fmul st(1), st

fsub qword ptr[decr]

loop faclup

}

endfuc:

\_\_asm{

fld st(1)

fld st(3)

fdiv st, st(1)

mov eax, i

test eax, 1

jne calculatesend

fchs

}

calculatesend:

\_\_asm {

fstp qword ptr[result\_buffer]

popad

}

result += result\_buffer;

}

//std::cout << "SResult: " << result << std::endl;

return result;

}

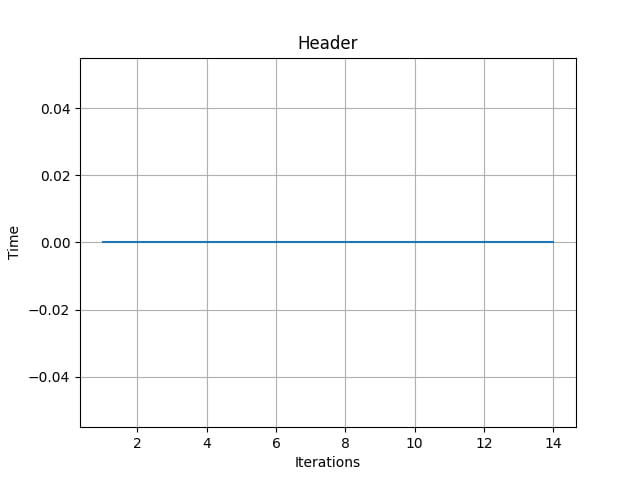
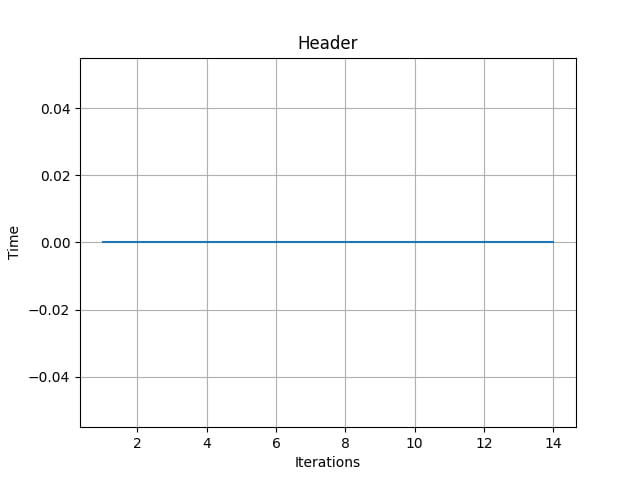


Рисунок 1 – Выходные данные с включенной SMT

Рисунок 2 – Выходные данные с выключенной SMT

**Выводы:** в результате лабораторной работы было выявлено, что в случае выполнения однопоточных вычислений при помощи математического сопроцессора процессорное время с включенной функцией SMT и с выключенной функцией SMT будет одинаковым.