ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)»

Кафедра МО ЭВМ

ОТЧЁТ

по лабораторной работе № 8

по дисциплине «Организация ЭВМ и систем»

Тема: Обработка вещественных чисел. Программирование математического сопроцессора.

Студент гр. 1303 _____ Ягодаров М.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

Разработать подпрограмму на языке Ассемблера, обеспечивающую вычисление заданной математической функции с использованием математического сопроцессора.

Задание.

Разработать подпрограмму на языке Ассемблера, обеспечивающую вычисление заданной математической функции с использованием математического сопроцессора. Подпрограмма должна вызываться из головной программы, разработанной на языке С. При этом должны быть обеспечены заданный способ вызова и обмен параметрами.

Выполнить трансляцию программы с подготовкой ее ассемблерной версии и отладочной информации. Для выбранного контрольного набора исходных данных прогнать программу под управлением отладчика. При этом для каждой команды сопроцессора следует фиксировать содержимое используемых ячеек памяти, регистров ЦП и численных регистров сопроцессора до и после выполнения этой команды.

Проверить корректность выполнения вычислений для нескольких наборов исходных данных.

Вариант 1.

Вернуть значение многочлена для заданного x.

Выполнение работы.

С помощью головной программы на языке Си производится считывание исходных данных: значения x, количества констант, а также из значения; значение x и констант принимается в виде чисел с плавающей запятой двойной точности.

После ввода числа даннные передаются в модуль, написанный на языке Ассемблера. В нём вычисляется значение многочлена с помощью схемы Горнера. На вход процедуре `poly` подаются данные: в регистре

xmm0 содержится x, в регистре rdi — количество констант, в регистре rsi — массив констант.

Первые 64 бита из xmm0 перемещается в регистр rax, содержимое которого после пушится на стек — число из стека считывается и записывается в стек математического сопроцессора с помощью инструкции `fld`, в математический стек также кладётся ноль. Далее проверяется количество констант: если их число равно нулю, то процедура завершается. Иначе в регистр rcx записывается значение rdi (число констант в массиве), после чего начинается цикл: значение верхушки стека математического сопроцессора умножается на второй элемент стека (индекс 1) с помощью инструкции `fmul` и операндом `st(1)`, затем к первому элементу стека прибавляется число из массива констант (инструкция `fadd`), получаемое обращением по адресу регистра rsi, смещённого на текущий номер константы, хранящийся в регистре rcx.

По итогу цикла в первом элементе стека математического сопроцессора будет содержаться результат вычисления многочлена; это число записывается в верхушку стека, которое после считывается и записывается в регистр rax. Значение из регистра rax записывается в первые 64 бита регистра xmm0 для того, чтобы соблюдать правила вызова функций для 64–разрядной операционной системы Linux.

В конце головная программа выводит значение многочлена в консоль.

Входные данные:

$$x = 1.7423$$

$$n = 3$$

$$1.8147, -2.8643, 0.5834$$

Таблица 1 – Результат прогона программы main в отладчике, начиная с момента вызова ассемблерного модуля.

Символический код команды	Содержимое регистров и ячеек памяти	
	До выполнения	После выполнения
movq rax, xmm0	rip = 0x5555555552ca rax = 0x3ffbe075f6fd21ff	rip = 0x5555555552cf rax = 0x3ffbe075f6fd21ff
push rax	rip = 0x5555555552cf rsp = 0x7fffffffdc78 Stack: +0 0x55555555527d	rip = 0x5555555552d0 rsp = 0x7fffffffdc70 Stack: +0 0x3ffbe075f6fd21ff +8 0x55555555527d
fld qword ptr [rsp]	rip = 0x5555555552d0 st0 = 0 fstat = 0x0 ftag = 0x0	rip = 0x5555555552d3 st0 = 1.74229999999999995985 fstat = 0x3800 ftag = 0x3fff
fldz	rip = 0x5555555552d3 st0 = 1.74229999999999995985 st1 = 0 fstat = 0x3800 ftag = 0x3fff	rip = 0x5555555552d5 st0 = 0 st1 = 1.74229999999999995985 fstat = 0x3000 ftag = 0x1fff
test rdi,rdi	rip = 0x5555555552d5 eflags = 0x246 [PF ZF IF]	rip = 0x5555555552d8 eflags = 0x206 [PF IF]
je poly_end	rip = 0x5555555552d8	rip = 0x5555555552da
mov rcx,rdi	rip = 0x5555555552da rcx = 0x555555559ac0	rip = 0x5555555552dd rcx = 0x3
fmul st,st(1)	rip = 0x5555555552dd st0 = 0 st1 = 1.74229999999999995985	rip = 0x5555555552df st0 = 0 st1 = 1.74229999999999995985

fadd qword ptr [rsi+rcx*8-8]	rip = 0x5555555552df st0 = 0 ftag = 0x1fff	rip = 0x5555555552e3 st0 = 0.583400000000000029665 ftag = 0xfff
loop horner	rip = 0x5555555552e3 rcx = 0x3	rip = 0x5555555552dd rcx = 0x2
fmul st,st(1)	rip = 0x5555555552dd st0 = 0.583400000000000029665 fstat = 0x3000	rip = 0x5555555552df st0 = 1.01645782000000002826 fstat = 0x3020
fadd qword ptr [rsi+rcx*8-8]	rip = 0x5555555552df st0 = 1.01645782000000002826	rip = 0x5555555552e3 st0 = -1.84784218000000003996
loop horner	rip = 0x5555555552e3 rcx = 0x3	rip = 0x5555555552dd rcx = 0x1
fmul st,st(1)	rip = 0x5555555552dd st0 = -1.84784218000000003996 fstat = 0x3020	rip = 0x5555555552df st0 = -3.21949543021399999551 fstat = 0x3220
fadd qword ptr [rsi+rcx*8-8]	rip = 0x5555555552df st0 = -3.21949543021399999551 fstat = 0x3220	rip = 0x5555555552e3 st0 = -1.40479543021400001576 fstat = 0x3020
loop horner	rip = 0x5555555552e3 rcx = 0x3	rip = 0x5555555552e5 rcx = 0x0
fstp qword ptr [rsp]	rip = 0x5555555552e5 Stack: +0 0x3ffbe075f6fd21ff	rip = 0x5555555552e8 Stack: +0 0xbff67a0ac5e56e1a

pop rax	rip = 0x5555555552e8 rax = 0x3ffbe075f6fd21ff rsp = 0x7fffffffdc70 Stack: +0 0xbff67a0ac5e56e1a +8 0x55555555527d	rip = 0x5555555552e9 rax = 0xbff67a0ac5e56e1a rsp = 0x7fffffffdc78 Stack: +0 0x55555555527d +8 0xa000000000000000
movq xmm0,rax	rip = 0x5555555552e9 xmm0 = 0x3ffbe075f6fd21ff	rip = 0x5555555552ee xmm0 = 0xbff67a0ac5e56e1a
ret	rip = 0x5555555552ee rsp = 0x7fffffffdc78 Stack: +0 0x55555555527d +8 0xa000000000000000	rip = 0x55555555527d rsp = 0x7fffffffdc80 Stack: +0 0xa000000000000000 +8 0x0000000300000003

Выводы

Получены навыки работы со специальными инструкциями Ассемблера для чисел с плавающей запятой. Разработана программа на ЯВУ Си, которая с помощью модуля написанного на языке Ассемблера и использованием математического сопроцессора, вычисляет, а после и отображает значение многочлена для заданного x.

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММ

Название файла: main.c

```
#include <stdio.h>
#include <stdlib.h>

extern double poly(double x, int n, double *c);

int main() {
 double x;
 printf("Enter x: ");
 scanf("%lf", &x);

 int n;
 printf("Enter number of constants: ");
 scanf("%d", &n);

 double *constants = malloc(n * sizeof(double));
 char c;
 printf("Enter constants: ");
 for (int i = 0; i < n; ++i) {
  scanf("%lf%c", &constants[i], &c);
 }

 double result = poly(x, n, constants);

 printf("(asm) Result is:\n\t%lf\n", result);
 free(constants);

 return 0;
}
```

Название файла: lib.s

```
.global poly

#; Input:
#; x: double  → xmm0
#; n: int     → rdi
#; c: double* → rsi
#; Output:
#; rax
poly:
 movq  rax, xmm0
 push  rax
 fld qword ptr [rsp]
 fldz
```

```
 test  rdi, rdi
 jz   poly_end
 mov  rcx, rdi
horner:
   fmul  st(1)
   fadd qword ptr [rsi + rcx * 8 - 8]
   loop  horner
poly_end:
 fstp qword ptr [rsp]
 pop  rax
 movq  xmm0, rax
 ret
```

Название файла: Makefile

```
all: main

main: main.o lib.o
	gcc main.o lib.o -o main -z noexecstack -lm

main.o: main.c
	gcc -c main.c

lib.o: lib.s
	as lib.s -msyntax=intel -mnaked-reg -mmnemonic=intel -o lib.o

clean:
	rm -f *.o main
```