Лабораторная работа №9

Программирование математического сопроцессора. Вызов ассемблерных функций из программ на C/C++

Цель работы:

Практическое овладение навыками разработки программного кода на языке Ассемблер. Изучить особенности программирования математического сопроцессора. Научиться стыковать модули, написанные с помощью языка Ассемблер и C++.

Постановка задачи

Разработка программы, использующей технологию совместного программирования на языках C/C++ и Ассемблер. В программе использовать арифметические операции сопроцессора.

1. Написать на базовом алгоритмическом языке C++ программу корректного ввода исходных данных (с контролем допустимого диапазона), вычисления арифметического выражения и вывода результата.
2. Написать модуль вычисления на языке Ассемблера.
3. Встроить вызов asm – модуля в программу на базовом алгоритмическом языке.
4. Произвести тестовые проверки и сделать анализ результатов (найдите значение выражения, используя MS Excel)

	A	B
1	x=	0,5
2	y=	0,5
3		
4	a=	=(1+(SIN(B1+B2))^2)/(2+ABS((B1-2*B1)/(1+B1*B1*B2*B2)))+B1*TAN(B1*B2)

В результате получилось:

	A	B
1	x=	0,5
2	y=	0,5
3		
4	a=	0,819034

5. Рассмотреть, как располагаются значения в стеке в процессе решения задачи.
6. В теоретической части вставить описание тех команд сопроцессора, которые были использованы в программе.

Содержание отчета:

1. Цель работы.
2. Постановка задачи.
3. Теоретическая часть (описание тех команд сопроцессора, которые были использованы в программах).
4. Листинги программ.
5. Пояснения к программе (таблицы изменения состояния стека сопроцессора).
6. Результаты работы программ +Excel.
7. Вывод.

Задание 1. Составить программу для вычисления значения функции

Вариант 1

$$b = \frac{\overline{2^{x^2 cosx^{8+1}}}}{2sinxlog_2 2sinx} - cos^2(x-1)$$

Вариант 2

$$b = \frac{\sqrt{2^{\sqrt{sinx+cosx}}}}{log_2 cos^2 x\sqrt{sinx * cosx}}$$

Вариант 3

$$b = \frac{2sinx * log_2\sqrt{x^4-1}}{\frac{\pi}{sin2x+\frac{\pi}{6}}};$$

Вариант 4

$$b = \frac{\sqrt{2^{x^2 cos\sqrt{x^8+1}}}}{2tgxlog_2 2tgx} - cos^2(x-1)$$

Вариант 5

$$b = \frac{\sqrt{2^{x^2 cos\sqrt{x^8+1}}}}{2sinxlog_2 2sinx} - tg\sqrt{x+1}$$

Вариант 6

$$b = \frac{2\sin x log_2\overline{x^4-1}+1}{\frac{2\pi}{sin2x+\frac{\pi}{6}}} - arctg^2(log_2\overline{x^4-1});$$

Вариант 7

$$b = \frac{2+2cos^2(x+2)}{\frac{tgx^4}{2+sin^2 x}} + 2\sqrt{sin(x^2+1)};$$

Вариант 8

$$a = \frac{2*\log_2\sqrt{x^4-1}}{\frac{\pi}{\sin x+\frac{\pi}{6}}} - tg^2(log_2\sqrt{sinx^2})$$

Вариант 9

$$b = \frac{1-cos^2(y+2)}{\frac{x^4}{2+sin^2 z}} + 2\sqrt{x^2+1};$$

Вариант 10

$$a = \frac{2}{tgx^2} + \frac{2^{xsin\sqrt{3x^2+1}}}{sinxcosxlog_2\sqrt{x^2+1}}$$

Вариант 11

$$b = \frac{2^{x\sin\sqrt{x^2+1}} - 1}{\cos x \sin x \log_2 \sqrt{x^2+1}} - arctg\sqrt{2^x};$$

Вариант 12

$$b = -\frac{1}{2} + artng\frac{\log_2 x}{\frac{1}{\sin^2 x + 2}} - \sin(\cos x^2);$$

Вариант 13

$$b = \frac{\sqrt{\sin^4 x + 1}}{2\frac{\cos x - 1}{\sin^2 x + 1}} - 2\sqrt{\sin x};$$

Вариант 14

$$b = \frac{2\sin x \log_2 \sqrt{x^4 - 1} + 1}{\frac{2\pi}{\sin 2x + \frac{\pi}{6}}} - arctg^2 x;$$

Вариант 15

$$b = \frac{\sqrt{2^{\sqrt{\sin x \cos x}}}}{\sqrt{\sin x \cos x}\log_2 \cos^2 x} - tg\sqrt{x+1}$$

Вариант 16

$$b = -1 + arctg\frac{z^2}{3 + \frac{x^2}{5}} + \sin\sqrt{x+1};$$

Вариант 17

$$b = \frac{\frac{2^{x\cdot\sin x^2+1} - 1}{tg^{\overline{x^2+1}} - 1}}{\sin x \cdot \log_2(\overline{x^2+1} + 1)} + tg\overline{2^x - 1}$$

Задание 2. Составьте программу вычисления значений функций

Вариант 1,3

$$f\ x = \begin{cases} \sin^2 x, \text{если } x \le -1 \\ 2^x, \text{если } -1 \le x \le 1 \\ 9x^2, \text{если } x \ge 1 \end{cases}$$

Вариант 2,5

$$f\ x = \begin{cases} \cos^2 x, \text{если } x < 0 \\ 2^x + 3, \text{если } 0 \le x \le 0{,}5 \\ tgx^2, \text{если } x > 0{,}5 \end{cases}$$

Вариант 4,7

$$f(x) = \begin{cases} \sin x + \cos x, \text{если } x < 1{,}1 \\ 2^x - 1, \text{если } 1{,}1 \le x \le 5{,}1 \\ 9x^2 + 9, \text{если } x > 5{,}1 \end{cases}$$

Вариант 6,9

$$f(x) = \begin{cases} \cos^2 x, \text{если } x < 0 \\ 2^x + 3, \text{если } 0 \le x \le 0{,}5 \\ arcgtx^3, \text{если } x > 0{,}5 \end{cases}$$

Вариант 8,11

$$f(x) = \begin{cases} \sin(\cos x), \text{если } x < 0 \\ 5\log_2(x+1), \text{если } 0 \le x \le 1{,}5 \\ cos(sinx), \text{если } x > 1{,}5 \end{cases}$$

Вариант 10,13

$$f(x) = \begin{cases} \sin x \cos x, \text{если } x < -0{,}9 \\ 2^x + 1, \text{если } -0{,}9 \le x \le 0{,}9 \\ 3x^3 - x^2 + 1, \text{если } x > 0{,}9 \end{cases}$$

Вариант 12,15

$$f(x) = \begin{cases} \sin^2 x, \text{если } x < 0 \\ 2^x + 7x^5, \text{если } 0 \le x \le 1{,}5 \\ x^3 + 2, \text{если } x > 1{,}5 \end{cases}$$

Вариант 14,17

$$f(x) = \begin{cases} \cos^4 x, \text{если } x < 0 \\ 2^x - 7, \text{если } 0 \le x \le 0{,}5 \\ (x^2+1)(x-1), \text{если } x > 0{,}5 \end{cases}$$

Вариант 15,19

$$f(x) = \begin{cases} \sin^4 x, \text{если } x < 0 \\ 2^x - 9, \text{если } 0 \le x \le 1 \\ x^3(x^2-1), \text{если } x > 1{,}5 \end{cases}$$

Теоретический материал

Сопроцессор

В процессорах Intel все операции с плавающей запятой выполняет специальное устройство, FPU (Floating Point Unit), с собственными регистрами и собственным набором команд, поставлявшееся сначала в виде сопроцессора (8087, 80287, 80387, 80487), а начиная с 80486DX — встраивающееся в основной процессор.

Числовой процессор может выполнять операции с семью разными типами данных: три целых двоичных, один целый десятичный и три типа данных с плавающей запятой.

Тип данных	Бит	Количество значащих цифр	Пределы
Целое слово	16	4	-32768 — 32767
Короткое целое	32	9	$-2*10^9$ — $2*10^9$
Длинное целое	64	18	$-9*10^{18}$ — $9*10^{18}$
Упакованное десятичное	80	18	-99..99 — +99..99 (18

			цифр)
Короткое вещественное	32	7	$1.18*10^{-38}$ — $3.40*10^{38}$
Длинное вещественное	64	15—16	$2.23*10^{-308}$ — $1.79*10^{308}$
Расширенное вещественное	80	19	$3.37*10^{-4932}$ — $1.18*10^{4932}$

Таблица 1. *Типы данных FPU*

Вещественные числа хранятся, как и все данные, в форме двоичных чисел. Двоичная запись числа с плавающей запятой аналогична десятичной, только позиции справа от запятой соответствуют не делению на 10 в соответствующей степени, а делению на 2.

Например, 0,625 = 0,101b. При записи вещественных чисел всегда выполняют нормализацию — умножают число на такую степень двойки, чтобы перед десятичной точкой стояла единица, в нашем случае $0,625 = 0,101b = 1,01b * 2^{-1}$.

Говорят, что число имеет мантиссу 1,01 и экспоненту -1. Как можно заметить, при использовании этого алгоритма первая цифра мантиссы всегда равна 1, так что ее можно не писать, увеличивая тем самым точность представления числа дополнительно на 1 бит. Кроме того, значение экспоненты хранят не в виде целого со знаком, а в виде суммы с некоторым числом так, чтобы хранить всегда только положительное число и чтобы было легко сравнивать вещественные числа — в большинстве случаев достаточно сравнить экспоненту. Теперь мы можем рассмотреть вещественные форматы IEEE, используемые в процессорах Intel:

- *короткое вещественное*: бит 31 — знак мантиссы, биты 30 – 23 — 8-битная экспонента + 127, биты 22 – 0 — 23-битная мантисса без первой цифры;
- *длинное вещественное*: бит 63 — знак мантиссы, биты 62 – 52 — 11-битная экспонента + 1024, биты 51 – 0 — 52-битная мантисса без первой цифры;
- *расширенное вещественное*: бит 79 — знак мантиссы, биты 78 – 64 — 15-битная экспонента + 16 383, биты 63 – 0 — 64-битная мантисса с первой цифрой (то есть бит 63 равен 1).

FPU выполняет все вычисления в 80-битном расширенном формате, а 32- и 64-битные числа используются для обмена данными с основным процессором и памятью.

Кроме обычных чисел формат IEEE предусматривает несколько специальных случаев, которые могут получаться в результате математических операций и над которыми также можно выполнять некоторые операции:

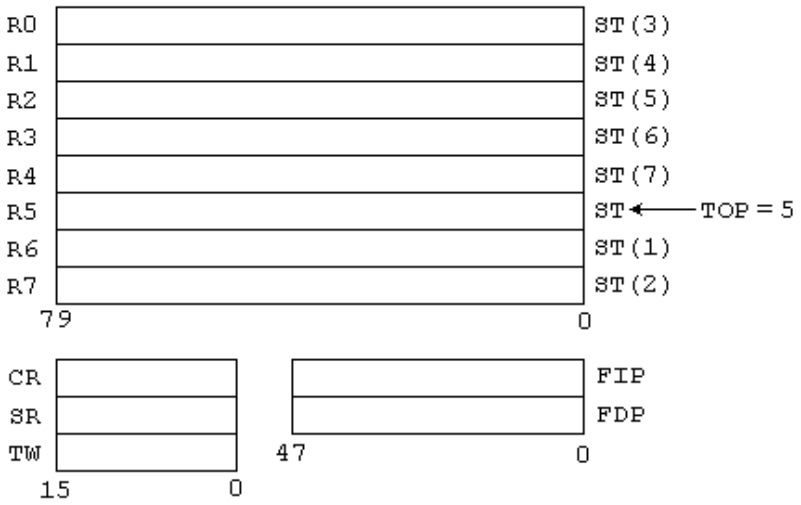
- *положительный ноль*: все биты числа сброшены в ноль;
- *отрицательный ноль*: знаковый бит — 1, все остальные биты — нули;
- *положительная бесконечность*: знаковый бит — 0, все биты мантиссы — 0, все биты экспоненты — 1;
- *отрицательная бесконечность*: знаковый бит — 1, все биты мантиссы — 0, все биты экспоненты — 1;
- *денормализованные числа*: все биты экспоненты — 0 (используются для работы с очень маленькими числами — до 10^{-16445} для расширенной точности);
- *неопределенность*: знаковый бит — 1, первый бит мантиссы (первые два для 80-битных чисел) — 1, а остальные — 0, все биты экспоненты — 1;
- *не-число типа SNAN (сигнальное)*: все биты экспоненты — 1, первый бит мантиссы — 0 (для 80-битных чисел первые два бита мантиссы — 10), а среди остальных бит есть единицы;

- *не-число типа QNAN (тихое)*: все биты экспоненты — 1, первый бит мантиссы (первые два для 80-битных чисел) — 1, среди остальных бит есть единицы. Неопределенность — один из вариантов QNAN;
- *неподдерживаемое число*: все остальные ситуации.

Остальные форматы данных FPU также допускают неопределенность — единица в старшем бите и нули в остальных для целых чисел, и старшие 16 бит — единицы для упакованных десятичных чисел.

FPU предоставляет восемь регистров для хранения данных и пять вспомогательных регистров.

Регистры данных (R0 – R7) не адресуются по именам, как регистры основного процессора. Вместо этого эти восемь регистров рассматриваются как стек, вершина которого называется ST, а более глубокие элементы — ST(1), ST(2) и так далее до ST(7). Если, например, в какой-то момент времени регистр R5 называется ST (рис. 13), то после записи в этот стек числа оно будет записано в регистр R4, который станет называться ST, R5 станет называться ST(1) и т.д.



Регистры FPU

Команды пересылки данных FPU

Команда	Тип данных	Характеристика операнда	Пример
Команды загрузки данных в стек			
FLD источник	Вещественное	Переменная в ОЗУ или ST(i),i=0,…7.	FLD a FLD ST(5)
FBLD источник	Типа BCD	Переменная в ОЗУ	FBLD aBc
FILD источник	Целое	Переменная в ОЗУ	FILD ka
Команды копирования данных из стека			
FST приемник	Вещественное	Переменная в ОЗУ или ST(i),i=1,…7.	FST ap FST ST(1)
FIST приемник	Целое	Переменная в ОЗУ	FIST kab
Команды извлечения данных из стека с освобождением вершины стека			
FSTP приемник	Вещественное	Переменная в ОЗУ или ST(i),i=1,…7.	FSTP app FSTP ST(5)

FBSTP приемник	Типа BCD	Переменная в ОЗУ	FBSTP abb
FISTP приемник	Целое	Переменная в ОЗУ	FISTP kabP
Команда обмена содержимого источника с ST(0)			
FXCH [источник]	Если источник НЕ указан, то считается, что он соответствует ST(0)	ST(i), i=0,…,7	FXCH FXCH ST(4)

Команды загрузки констант

Команда	Назначение
FLD1	Поместить в ST(0) число 1.0
FLDZ	Поместить в ST(0) число +0.0
FLDPI	Поместить в ST(0) число pi
FLDL2E	Поместить в ST(0) число, равное $\log_2 e$
FLDL2T	Поместить в ST(0) число, равное $\log_2 10$
FLDLN2	Поместить в ST(0) число, равное ln 2
FLDLG2	Поместить в ST(0) число, равное lg 2

Арифметические команды

Команды базовой арифметики:

Команда	Тип данных	Алгоритм выполнения
Команды сложения		
FADD приемник, источник	Вещественное	Приемник = Приемник + Источник
FADDP приемник, источник	Вещественное	
FIADD источник	Целое	
Команды обычного вычитания		
FSUB приемник, источник	Вещественное	Приемник = Приемник - Источник
FSUBP приемник, источник	Вещественное	
FISUB источник	Целое	
Команды реверсного (обратного) вычитания		
FSUBR приемник, источник	Вещественное	Приемник = Источник - Приемник
FSUBRP приемник, источник	Вещественное	
FISUBR источник	Целое	
Команды умножения		
FMUL приемник, источник	Вещественное	Приемник = Приемник * Источник
FMULP приемник, источник	Вещественное	
FIMAL источник	Целое	
Команды обычного деления		
FDIV приемник, источник	Вещественное	Приемник = Приемник / Источник
FDIVP приемник, источник	Вещественное	
FIDIV источник	Целое	
Команды реверсного (обратного) деления		

FDIVR приемник, источник	Вещественное	Приемник = Источник / Приемник
FDIVRP приемник, источник	Вещественное	
FIDIVR источник	Целое	

Специальные арифметические команды:

Команда	Назначение	Алгоритм выполнения
FPREM	Нахождение частичного остатка от деления путем последовательного вычитания 64 раза содержимого ST(1) из ST(0)	ST(0) = остаток [ST(0) / ST(1)] Формируется флаг C2 регистра управления CWR: C2 = 0 - получен точный остаток от деления, т.е. остаток < ST(1). C2 = 1 - точный остаток НЕ получен (получен частичный остаток)
FPREM (i80387)	Нахождение частичного остатка от деления в стандарте IEEE - округление к ближайшему целому	
FABS	Нахождение абсолютного значения	ST(0) = abs (ST(0))
FSQRT	Нахождение корня квадратного	ST(0) = sqrt (ST(0))
FCHS	Изменение знака	ST(0) = - ST(0)
FRNDINT	Округление до целого	Содержимое ST(0) округляется до целого в соответствии с битами RC регистра управления сопроцессором CWR
FXTRACT	Извлечь экспоненту и мантиссу числа	Число ===> ST(0) ST(0) = мантисса числа; ST(1) = экспонента числа
FSCALE	Команда, обратная FXTRACT	ST(0) <=== мантисса числа; ST(1) <=== экспонента числа; ST(0) = ST(0)*$2^{ST(1)}$

Трансцендентные операции

Команда	Назначение	Алгоритм выполнения
FSIN (i80387)	Вычисление синуса sin(x), где значение аргумента x задается в радианах	x ===>ST(0); ST(0)=sin(ST(0)); 2^{63}<=x<=2^{63}. Если значение x выходит за эти пределы, можно воспользоваться командой FPREM с делителем 2π. Иначе флаг C2=1 и значение ST(0) не изменяется.
FCOS (i80387)	Вычисление косинуса cos(x), где значение аргумента x задается в радианах	x ===>ST(0); ST(0)=cos(ST(0)); 2^{63}<=x<=2^{63}. Если значение x выходит за эти пределы, можно воспользоваться командой FPREM с делителем 2π. Иначе флаг C2=1 и значение ST(0) не изменяется.

Команда	Назначение	Алгоритм выполнения
FSINCOS (i80387)	Вычисление синуса и косинуса угла	x ===>ST(0); ST(1)=sin (ST(0)); ST(0)=cos(ST(0)); 2^{63}<=x<=2^{63}. Если значение x выходит за эти пределы, можно воспользоваться командой FPREM с делителем 2π. Иначе флаг C2=1 и значение ST(0) не изменяется.
FPTAN	Вычисление частичного тангенса tg(a)=y/x, где значение аргумента a задается в радианах	a===>ST(0); ST(0)=x; ST(1)=y; 0<a<π/4 (i8087, i80287). 2^{63}<=a<=2^{63}, (начиная с i80387). Если значение a выходит за эти пределы, можно воспользоваться командой FPREM с делителем 2П. Иначе флаг C2=1 и значение ST(0) не изменяется.
FPATAN	Вычисление арктангенса a = arctg(y/x)	ST(0)=x; ST(1)=y; a<==ST(0), знак совпадает со знаком ST(1). 0 < |y| < |x| < бесконечность. |a|<π
F2XMI	Вычисление 2x-1	X===>ST(0); ST(0)=2^x-1. -1 < X < 1, иначе результат операции не определен
FYL2X	Вычисление y*log2(x)	x===>ST(0); y===>ST(1); 0 < x < бесконечность, -бесконечность < y < +бесконечность
FYL2XPI	Вычисление y*log2(c+1)	c===>ST(0); y===>ST(1); 0 <= |c| < 1-$2^{0.5}$/2. -бесконечность < y < +бесконечность

Команды сравнения

Команда	Назначение
FCOM источник	Сравнение вещественных данных
FCOMP источник	Сравнение вещественных данных и освобождение вершины стека ST(0)
FCOMPP	Сравнение вещественных данных и освобождение вершины стека ST(0) и регистра ST(1)
FUCOM источник (i80387)	Сравнение вещественных данных БЕЗ учета порядков
FUCOMP источник (i80387)	Сравнение вещественных данных БЕЗ учета порядков и освобождение вершины стека ST(0)
FUCOMPP (i80387)	Сранение вещественных данных БЕЗ учета порядков и освобождение вершины стека ST(0) и регистра ST(1)
FICOM источник	Сравнение целочисленных данных
FICOMP источник	Сравнение целочисленных данных и освобождение вершины стека ST(0)

Команда	Назначение
FCOMI ST(0), ST(1) (Pentium Pro)	Сравнение данных и установка EFLAGS
FCOMIP ST(0), ST(1) (Pentium Pro)	Сравнение данных, установка EFLAGS и освобождение вершины стека ST(0)
FUCOMI ST(0), ST(1) (Pentium Pro)	Сравнение данных БЕЗ учета порядков и установка EFLAGS
FUCOMIP ST(0), ST(1) (Pentium Pro)	Сравнение данных БЕЗ учета порядков, установка EFLAGS и освобождение вершины стека ST(0)
FTST	Сравнение содержимого ST(0) с нулем
FXAM	Анализ содержимого и определение типа числа в ST(0)

Флаги сравнения для команд базового сопроцессора:

Условие	Флаг C3	Флаг C2	Флаг C0
ST(0) > источник	0	0	0
ST(0) < источник	0	0	1
ST(0) = источник	1	0	0
Операнды несравнимы	1	1	1

Флаги сравнения для команд сопроцессора Pentium Pro:

Условие	Флаг ZF	Флаг PF	Флаг CF
ST(0) > источник	0	0	0
ST(0) < источник	0	0	1
ST(0) = источник	1	0	0
Операнды несравнимы	1	1	1

Благодаря тому, что сразу формируются флаги EFLAGS процессора, команды сравнения для сопроцессора Pentium Pro работают быстрее и не изменяют содержимого регистра AX, но НЕ все сопроцессоры сторонних от Intel производителей (и компиляторы) их поддерживают.

Результат действия команды FXAM:

Тип числа в ST(0)	Флаг C3	Флаг C2	Флаг C0
Не поддерживаемое	0	0	0
Не - число	0	0	1
Нормальное конечное число	0	1	0
Бесконечность	0	1	1
Нуль	1	0	0
Регистр пуст	1	0	1
Денормализованное число	1	1	0

Флаг C1 устанавливается равным знаку числа в ST(0)

Команды управления сопроцессором

Команда	Назначение
FLDCW источник	Загрузка регистра управления CWR из источника (слово в оперативной памяти)
FSTCW приемник	Запись содержимого слова управления CWR в приемник (оперативная память). Данная команда полностью эквивалентна команде WAIT FNSTCW
FNSTCW приемник	Запись содержимого слова управления CWR в приемник (оперативная память) без ожидания окончания обработки исключительных ситуаций
FSTSW приемник (i80287)	Запись содержимого слова состояния SWR в приемник (оперативная память). Данная команда полностью эквивалентна команде WAIT FNSTSW
FNSTSW приемник (i80287)	Запись содержимого слова состояния SWR в приемник (оперативная память) без ожидания окончания обработки исключительных ситуаций
FSAVE приемник	Сохранение полного состояния FPU (регистров данных и вспомогательных регистров)в приемнике (участок памяти размером 94 или 108 байт). Данная команда полностью эквивалентна команде WAIT FNSAVE
FNSAVE приемник	Сохранение полного состояния FPU (регистров данных и вспомогательных регистров) без ожидания окончания обработки исключительных ситуаций
FXSAVE приемник (Pentium II)	Быстрое восстановление полного состояния FPU (регистров данных и вспомогательных регистров) в приемнике (участок памяти размером 512 байт). Данная команда не совместима с командами FSAVE/FRSTOP.
FRSTOP источник	Восстановление полного состояния FPU (регистров данных и вспомогательных регистров). Данная команда является обратной команде FSAVE.
FXSTOP источник (Pentium II)	Быстрое восстановление полного состояния FPU (регистров данных и вспомогательных регистров). Данная команда является обратной команде FXSAVE.
FLDENV источник	Загрузка из источника (14 или 28 байт в памяти в зависимости от разрядности операндов) пяти вспомогательных регистров окружения сопроцессора (CWR, SWR, TWR, FIP, FDP).
FSTENV источник	Сохранение пяти вспомогательных регистров. Данная команда полностью эквивалентна команде WAIT FNSTENV и является обратной команде FLDENV
FNSTENV источник	Сохранение пяти вспомогательных регистров без ожидания окончания обработки исключительных ситуаций. Данная команда является обратной команде FLDENV.
FWAIT WAIT	Ожидание готовности сопроцессора
FINIT	Инициализация сопроцессора. Данная команда полностью эквивалентна команде WAIT FNINIT
FNINIT	Инициализация сопроцессора без ожидания окончания обработки исключительных ситуаций.
FENI (только в i8087)	Включение исключений. В старших версиях сопроцессора воспринимается как команда FNOP
FDISI (только в i8087)	Запрещение исключений. В старших версиях сопроцессора воспринимается как команда FNOP
FNOP	Отсутствие операции
FCLEX	Обнуление флагов исключений в регистре состояния SWR (PE, UE, OE, ZE, DE, IE, ES, SE и B). Данная команда полностью эквивалента команде WAIT FNCLEX.
FNCLEX	Обнуление флагов исключений без ожидания.
FINCSTP	Поле TOP регистра состояния сопроцессора увеличивается на 1.
FDECSTP	Поле TOP регистра состояния сопроцессора уменьшается на 1.
FFREE ST(i)	Освобождение регитра данных ST(i), i=0,…,7. В регистре тегов TWR данный регистр помечается как пустой.

Стыковка модулей

Стыковка модулей Ассемблера с модулями, написанными на языках C/C++ требует выполнения следующих условий:

Требования к главному модулю на языке C++:

1. Имена внешних переменных и функций должны быть описаны, как глобальные.
2. Имена внешних функций должны быть уникальными.
3. К именам внешних функций должен быть применен спецификатор сборки **extern "C" {***прототипы_функций***}**, чтобы они интерпретировались в стиле языка C.
4. Константы внешними быть не могут.
5. Вызов внешних функций делается, как обычно это принято в языке C++.
6. **Стыкуемые модули должны быть описаны в проекте.** Поскольку мы сейчас изучаем 16-разрядное программирование на Ассемблере, проект должен быть ТОЛЬКО **Application [.exe] | DOS (standard).**

Требования к модулю на языке Ассемблера:

1. Модель желательно делать **LARGE** (во избежание проблем с оперативной памятью, особенно в 32- и 64-разрядных операционных системах) с ОБЯЗАТЕЛЬНЫМ указанием языка программирования **C:**

   ```
   model large, C
   ```

2. **Внешние переменные,** передаваемые из CPP – модуля, **могут быть описаны как в сегменте данных, так и в сегменте кода** с помощью директивы EXTERN.
3. **Описание констант должно быть ПОВТОРЕНО.** Они должны быть ЛОКАЛЬНЫМИ и могут быть расположены как в сегменте данных, так и в сегменте кода.
4. Внешние функции должны быть описаны с помощью директивы описания общих имен с ОБЯЗАТЕЛЬНЫМ указанием языка программирования C:

   PUBLIC C *имя_внешней_функции*

5. Описание сегментов лучше делать так : **.Code** и **.Data**

Пример: вычислить значение выражения x = (2*a + b*c)/(d-a).

primC.cpp

```
#include <conio.h>
#include <iostream.h>
#include <limits.h>

inline int test (long int a)
     {return ((a>>15)+1)&~1;}

int primC(int a, const int b, const int c, const int d)
{ double z=(2.0*a+1.0*b*c)/(d-a);
  if (z > SHRT_MIN && z < SHRT_MAX) return z;
  else
  {cout << "\n!!!!! Limits of int value !!!!!!!\n x=" << z << endl;
   return SHRT_MIN; //-32768
```

```
  }
}

extern "C"
{ void prim (void);}  //внешняя функция, которая будет написана на
                      // Ассемблере

int X, a  //глобальные переменные

void main (void)
{
      char ch;
      const b=-333;
      const c=1000;
      const d=-10;
      long int a1;
      do
      {
          X = 0;
          cout << "\n x=(2*a+b*c)/(d-a);
          int x, a, b = -333, c = 1000,d = -10;" << endl;
          do
          {
           cout << "\n Enter a [-32768…32767], a!="<< d << "====>";
           cin >> a1;
          }
          while (test (a1) || d - a1 ==0 || test (d-a1));
          a=a1;
          X=primC(a, b, c, d);
          if (X!=SHRT_MIN)
          {cout << "Result (C++) x = " << X << endl;
           X=0;
           prim();
           cout << " Result (ASM) x = " << X << endl;
          }
           cout << "\n\nExit? - (y/n)\n";
           ch=getch();
          }
          while (!(ch=='y' || ch=='Y'));
}
```

Prim.asm

```
title Арифметические выражения
model large, C
CODESEG ; начало сегмента кода
;======== ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ==========
      Extrn C X:word
      Extrn C a:word
;локальные перменные - константы
      b dw -333
      c dw 1000
      d dw -10
;============================
      Public C prim
prim proc far
      mov ax, 2
```

```
      Imul a       ; <dx>:<ax>=2*a
      mov bx, dx    ; bx <====ст.часть (dx)
      mov cx, ax    ;cx <====мл. часть (ax)
      mov ax, b
      Imul c          ; <dx>:<ax>=b*c
      add ax, cx     ;<ax>=<ax>+<cx>  (мл.часть)
      adc dx, bx     ;<dx> =<dx>+<bx>  (ст.часть)
      mov cx, d
      sub cx, a       ;<cx>=<cx>-a
      Idiv cx           ;<ax>=<dx>:<ax>/<cx>
      mov X, ax
      ret
prim endp
end
```

Далее необходимо просто скомпилировать файл проекта.
Если все действия были сделаны верно, то компилятор автоматически добавит файл prim.asm к проекту.
Во избежание ошибок рекомендуется все файлы сохранять в папке \bc5\bin\. Туда же рекомендуется поместить файл tasm.exe.
Если не получается присоединить автоматически файл Ассемблера, то необходимо его откомпилировать вручную (т.е. получить файл с расширением .obj.
Например: tasm prim.asm /l /ml), а затем уже самостоятельно добавить к проекту, и еще раз откомпилировать уже в среде BC++.
ВНИМАНИЕ: ключ компиляции в этом случае /ml, что указывает на необходимость учета регистра букв.

Пример работы сопроцессора:

```
.8087
data segment para
      EXTRN a5:Dword, c5:word, d5:word, y:Dword
data ends

code segment para
      assume cs:code, ds:data
      public var5
      four dd 4
      s dd 62
      one dd 1.0

var5 proc FAR
      finit
      fld one              (ST(0)=1.0)
      fld a5               (ST(0)=a5, ST(1)=1)
      fmul st(0), st(0)    (ST(0)=a5*a5, ST(1)=1.0)
      fadd st(0), st(1)    (ST(0)=a5*a5+1, ST(1)=1.0)
      fild c5              (ST(0)=c5, ST(1)=a5*a5+1, ST(2)=1.0)
      fild four            (ST(0)=4, ST(1)=c5, ST(2)=a5*a5+1, ST(3)=1.0)
      fpatan               (ST(0)=arctg(c5/4), ST(1)=a5*a5+1, ST(2)=1.0)
      fst y
      ret
var5 endp
code ends
end
```

Пример 2. Дано действительное число a. Вычислить f(a), если

$$f(x) = \begin{cases} a+1, \text{если } a \geq 0; \\ a, \text{если } a < 0. \end{cases}$$

```
ассемблерная вставка
__asm
{
fld1
fldz
fld a
fcom st(1)
fstsw ax
sahf
jb l1
fadd st(0), st(1)
jmp p1
l1:
fabs
p1: fstp a
…
```