МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

#### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ

БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет автоматики и вычислительной техники

Кафедра радиоэлектронных средств

Отчет по дисциплине

«Цифровые устройства и микропроцессоры»

Лабораторная работа №3

«ПРИНЦИПЫ

«ИСПОЛЬЗОВАНИЕ МАТЕМАТИЧЕСКОГО СОПРОЦЕССОРА»

Вариант №15

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИНБс-3301-01-00 | \_\_\_\_\_\_\_\_\_\_\_\_ | С.А. Симахин |
| Проверил: доцент кафедры РЭС | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | М. А. Земцов |

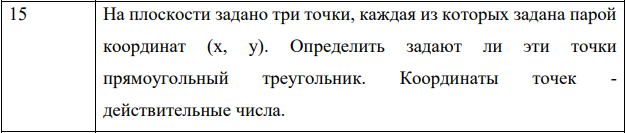
#### Киров 2025

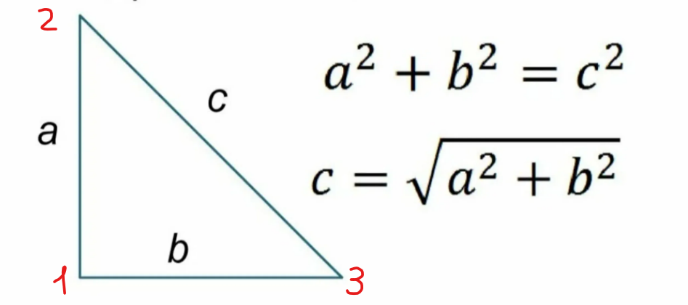
**Цель работы:** изучение принципов выполнения арифметических команд с помощью математического сопроцессора FPU микропроцессоров с архитектурой x86.

**Ход работы:**

Исходные данные представлены в таблице 1 и пояснении ниже.

Таблица 1 – Исходные данные



Для проверки, является ли фигура прямоугольным треугольником можно использовать теорему Пифагора:

a – между точками 2 и 1

b – между точками 3 и 1

c – между точками 3 и 2

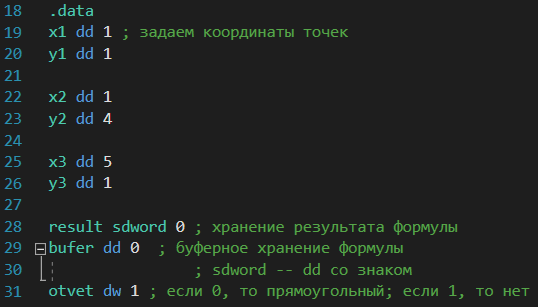
Пример:

Для нахождения расстояния между точками 2 и 1 (длина а) можно использовать формулу:

Преобразуя теорему Пифагора в координатный вид, получим:

Далее упростим формулу, получим конечный вид проверки, которая используется в коде:

Далее начинаем разбор кода:



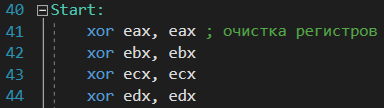
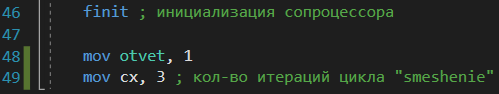
Рисунок 1 – Задаем координаты точек треугольника



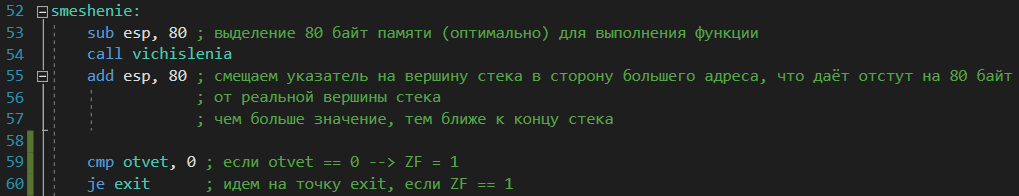
Рисунок 2 – очистка регистров



46 строка: командой **finit** производим инициализацию сопроцессора с плавающей запятой (FPU), также данная команда выполняет:  
 - сброс всех управляющих регистров   
 - сброс параметров точности вычислений и округлений  
 - очистка стека FPU

****48 строка: ввод 1 в переменную **otvet**

49 строка: количество итераций для цикла **smeshenie**



52 строка: начало цикла **smeshenie**

53 строка: ввод 80 в регистр **esp** (пояснения не скрине), **можно посмотреть в памяти, поиск: esp, после каждой итерации требуется обновление поиска**

54 строка: вызов функции **vichislenia** (разбор функции ниже)

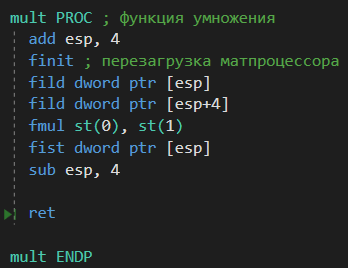
55 строка: прибавление 80 к значению в регистрe **esp** (пояснения на скрине)

59 строка: сравнение значения переменной **otvet** с **0**

60 строка: переход на точку **exit**, если флаг ZF = 1

Для последующего понимания кода, необходимо заранее разобрать функции:

1. Функция умножения:



**add esp, 4** – добавление в стек 4 байта

**finit** – перезагрузка сопроцессора для сброса всех регистров прошлых операций

**fild dword ptr [esp]** – загрузка целого числа **esp** в стек

**fild dword ptr [esp+4]** – загрузка целого числа **esp + 4 байта** в стек

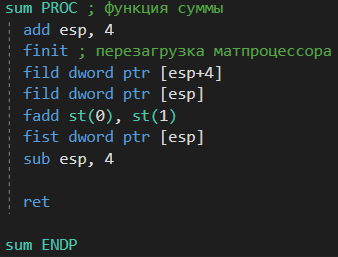
**fmul st(0), st(1) –** перемножение значений **st(0)** и **st(1)**, результат в **st(0)**

**fild dword ptr [esp]** – загрузка целого числа **esp** в стек

**sub esp, 4 –** возврат адресации на 4 байта

**ret –** возвращение в место вызова функции

1. Функция суммы:



**add esp, 4** – добавление в стек 4 байта

**finit** – перезагрузка сопроцессора для сброса всех регистров прошлых операций

**fild dword ptr [esp+4]** – загрузка целого числа **esp+ 4 байта** в стек

**fild dword ptr [esp]** – загрузка целого числа **esp** в стек

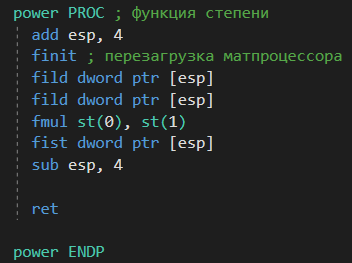
**fadd st(0), st(1) –** сумма значений **st(0)** и **st(1)**, результат в **st(0)**

**fild dword ptr [esp]** – загрузка целого числа **esp** в стек

**sub esp, 4 –** возврат адресации на 4 байта

**ret –** возвращение в место вызова функции

1. Функция степени:



**add esp, 4** – добавление в стек 4 байта

**finit** – перезагрузка сопроцессора для сброса всех регистров прошлых операций

**fild dword ptr [esp]** – загрузка целого числа **esp** в стек

**fild dword ptr [esp]** – загрузка целого числа **esp** в стек

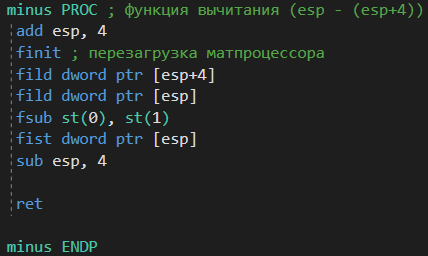
**fmul st(0), st(1) –** перемножение значений **st(0)** и **st(1)**, результат в **st(0)**

**fild dword ptr [esp]** – загрузка целого числа **esp** в стек

**sub esp, 4 –** возврат адресации на 4 байта

**ret –** возвращение в место вызова функции

1. Функция вычитания:



**add esp, 4** – добавление в стек 4 байта

**finit** – перезагрузка сопроцессора для сброса всех регистров прошлых операций

**fild dword ptr [esp+4]** – загрузка целого числа **esp + 4 байта** в стек

**fild dword ptr [esp]** – загрузка целого числа **esp** в стек

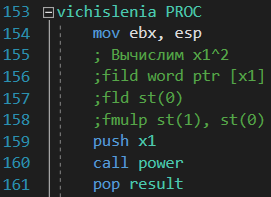
**fsub st(0), st(1) –** вычитание значений **st(0)** и **st(1)**, результат в **st(0)**

**fild dword ptr [esp]** – загрузка целого числа **esp** в стек

**sub esp, 4 –** возврат адресации на 4 байта

**ret –** возвращение в место вызова функции

**Разбор функции vichislenia (153 строка – 278 строка):**



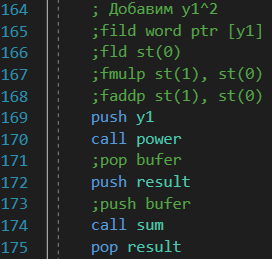
****153 строка: начало функции **vichislenia**

154 строка: ввод в регистр **ebx** значения регистра **esp**

159 строка: ввод в стек значения **x1**

160 строка: вызов функции **power** для возведения числа в степень

161 строка: выгрузка из стека результата в переменную **result**



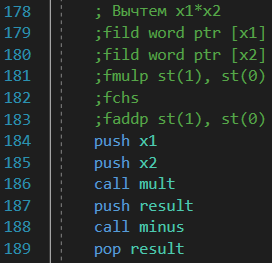
169 строка: ввод в стек значения **y1**

170 строка: вызов функции **power** для возведения числа в степень

172 строка: ввод в стек значения переменной **result**

174 строка: вызов функции **sum** для вычисления суммы чисел

175 строка: выгрузка из стека результата в переменную **result**



184 строка: ввод в стек значения **x1**

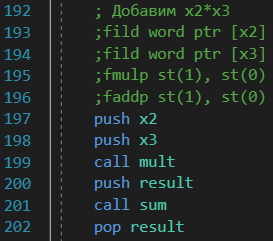
185 строка: ввод в стек значения **x2**

186 строка: вызов функции **mult** для перемножения значений

187 строка: ввод в стек значения переменной **result**

188 строка: вызов функции **minus** для вычитания одного числа из другого

189 строка: выгрузка из стека результата в переменную **result**



197 строка: ввод в стек значения **x2**

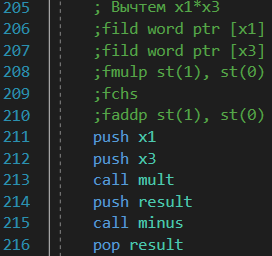
198 строка: ввод в стек значения **x3**

199 строка: вызов функции **mult** для перемножения значений

200 строка: ввод в стек значения переменной **result**

201 строка: вызов функции **sum** для вычисления суммы чисел

202 строка: выгрузка из стека результата в переменную **result**



211 строка: ввод в стек значения **x1**

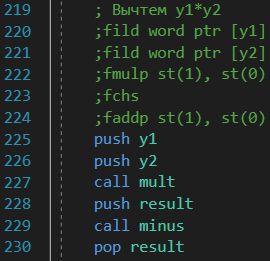
212 строка: ввод в стек значения **x3**

213 строка: вызов функции **mult** для перемножения значений

214 строка: ввод в стек значения переменной **result**

215 строка: вызов функции **minus** для вычитания одного числа из другого

216 строка: выгрузка из стека результата в переменную **result**



225 строка: ввод в стек значения **y1**

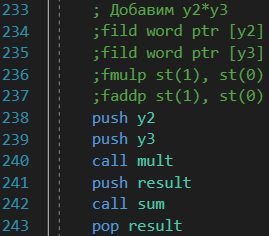
226 строка: ввод в стек значения **y2**

227 строка: вызов функции **mult** для перемножения значений

228 строка: ввод в стек значения переменной **result**

229 строка: вызов функции **minus** для вычитания одного числа из другого

230 строка: выгрузка из стека результата в переменную **result**



238 строка: ввод в стек значения **y2**

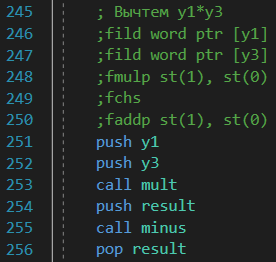
239 строка: ввод в стек значения **y3**

240 строка: вызов функции **mult** для перемножения значений

241 строка: ввод в стек значения переменной **result**

242 строка: вызов функции **sum** для вычисления суммы чисел

243 строка: выгрузка из стека результата в переменную **result**



251 строка: ввод в стек значения **y1**

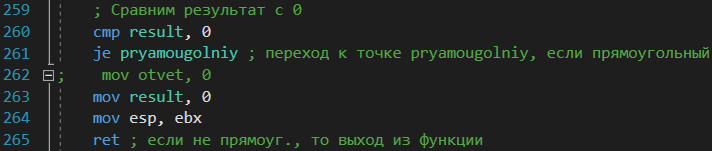
252 строка: ввод в стек значения **y3**

253 строка: вызов функции **mult** для перемножения значений

254 строка: ввод в стек значения переменной **result**

255 строка: вызов функции **minus** для вычитания одного числа из другого

256 строка: выгрузка из стека результата в переменную **result**



260 строка: сравнивание значения переменной **result** c **0** если совпадает, то

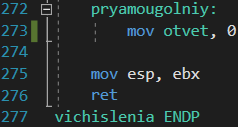
поднимается флаг ZF(ZR) = 1

261 строка: переход в точку **pryamougolniy**(269 строка), если флаг ZF(ZR) = 1

263 строка: ввод в переменную **result** значения **0**

264 строка: ввод в регистр **esp** значения регистра **ebx**

265 строка: возвращение в место вызова функции



272 строка: вход в точку **pryamougolniy**

273 строка: ввод в переменную **otvet** значения **0**

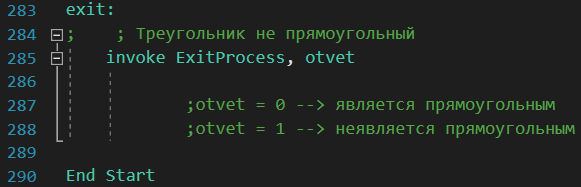
275 строка: ввод в регистр **esp** значения регистра **ebx**

276 строка: возвращение в место вызова функции

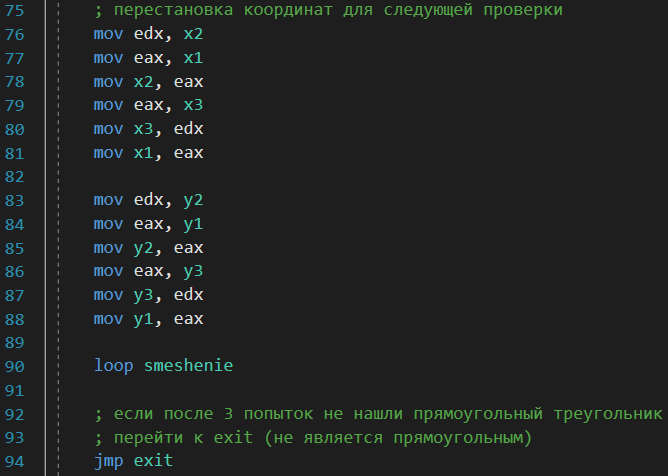
277 строка: конец функции **vichislenia**

После прохождения функции **vichislenia** идет смещение указателя на вершину стека, в сторону большего адреса (на 80 байт) (55 строка)

Далее идет ключевая проверка, если значение в переменной равно 0, т.е. выполняется равенство проверки является ли треугольник прямоугольным, то переходим на точку **exit** и выходим из программы, возвращая данный результат в консоль (285 строка)

Если значение не удовлетворяет условию, то мы производим перестановку координат, для последующей проверки. Если после 3 циклов итераций равенство так и не было равно 0 🡪 прыгаем на выход программы (94 строка) и возвращаем 1, что обозначает, что треугольник не является прямоугольным



Код программы:

; 15 вариант

; На плоскости задано три точки, каждая из которых задана парой

; координат (x, y). Определить задают ли эти точки

; прямоугольный треугольник. Координаты точек -

; действительные числа.

; для проверки, является ли фигура прямоугольным треугольником, можно использовать теорему Пифагора

;

; Формула, при истинности которой объект является прямоугольным треугольником:

; x1^2+y1^2-x1x2+x2x3-x1x3-y1y2+y2y3-y1y3 = 0

.686

.model flat, stdcall

.stack 100h

.data

x1 dd 1 ; задаем координаты точек

y1 dd 1

x2 dd 1

y2 dd 4

x3 dd 5

y3 dd 1

result sdword 0 ; хранение результата формулы

bufer dd 0 ; буферное хранение формулы

; sdword -- dd со знаком

otvet dw 1 ; если 0, то прямоугольный; если 1, то нет

.code

ExitProcess PROTO STDCALL :DWORD

; последняя догадка: передавать функции через cтек, и обращаться в нужноее место стека для инициализации того, какой блок и с какими переменными программа будет выполняться

Start:

xor eax, eax ; очистка регистров

xor ebx, ebx

xor ecx, ecx

xor edx, edx

finit ; инициализация сопроцессора

mov otvet, 1

mov cx, 3 ; кол-во итераций цикла "smeshenie"

smeshenie:

sub esp, 80 ; выделение 80 байт памяти (оптимально) для выполнения функции

call vichislenia

add esp, 80 ; смещаем указатель на вершину стека в сторону большего адреса, что даёт отстут на 80 байт

; от реальной вершины стека

; чем больше значение, тем ближе к концу стека

cmp otvet, 0 ; если otvet == 0 --> ZF = 1

je exit ; идем на точку exit, если ZF == 1

; после возврата из функции проверим ZF (флаг нуля)

; если ZF=1, значит прямоугольный треугольник

; если нет, меняем координаты и повторяем

; проверка флага ZF после vichislenia

; vichislenia должна устанавливать флаг ZF через sahf

; Проверим флаг ZF

; sahf загружает флаги из AH, поэтому флаг ZF доступен напрямую

; если ZF=1, перейти к obhod

;jz obhod

; перестановка координат для следующей проверки

mov edx, x2

mov eax, x1

mov x2, eax

mov eax, x3

mov x3, edx

mov x1, eax

mov edx, y2

mov eax, y1

mov y2, eax

mov eax, y3

mov y3, edx

mov y1, eax

loop smeshenie

; если после 3 попыток не нашли прямоугольный треугольник

; перейти к exit (не является прямоугольным)

jmp exit

mult PROC ; функция умножения

add esp, 4

finit ; перезагрузка матпроцессора

fild dword ptr [esp]

fild dword ptr [esp+4]

fmul st(0), st(1)

fist dword ptr [esp]

sub esp, 4

ret

mult ENDP

sum PROC ; функция суммы

add esp, 4

finit ; перезагрузка матпроцессора

fild dword ptr [esp+4]

fild dword ptr [esp]

fadd st(0), st(1)

fist dword ptr [esp]

sub esp, 4

ret

sum ENDP

power PROC ; функция степени

add esp, 4

finit ; перезагрузка матпроцессора

fild dword ptr [esp]

fild dword ptr [esp]

fmul st(0), st(1)

fist dword ptr [esp]

sub esp, 4

ret

power ENDP

; \*выстрел\*

minus PROC ; функция вычитания (esp - (esp+4))

add esp, 4

finit ; перезагрузка матпроцессора

fild dword ptr [esp+4]

fild dword ptr [esp]

fsub st(0), st(1)

fist dword ptr [esp]

sub esp, 4

ret

minus ENDP

vichislenia PROC

mov ebx, esp

; Вычислим x1^2

;fild word ptr [x1]

;fld st(0)

;fmulp st(1), st(0)

push x1

call power

pop result

; Добавим y1^2

;fild word ptr [y1]

;fld st(0)

;fmulp st(1), st(0)

;faddp st(1), st(0)

push y1

call power

;pop bufer

push result

;push bufer

call sum

pop result

; Вычтем x1\*x2

;fild word ptr [x1]

;fild word ptr [x2]

;fmulp st(1), st(0)

;fchs

;faddp st(1), st(0)

push x1

push x2

call mult

push result

call minus

pop result

; Добавим x2\*x3

;fild word ptr [x2]

;fild word ptr [x3]

;fmulp st(1), st(0)

;faddp st(1), st(0)

push x2

push x3

call mult

push result

call sum

pop result

; Вычтем x1\*x3

;fild word ptr [x1]

;fild word ptr [x3]

;fmulp st(1), st(0)

;fchs

;faddp st(1), st(0)

push x1

push x3

call mult

push result

call minus

pop result

; Вычтем y1\*y2

;fild word ptr [y1]

;fild word ptr [y2]

;fmulp st(1), st(0)

;fchs

;faddp st(1), st(0)

push y1

push y2

call mult

push result

call minus

pop result

; Добавим y2\*y3

;fild word ptr [y2]

;fild word ptr [y3]

;fmulp st(1), st(0)

;faddp st(1), st(0)

push y2

push y3

call mult

push result

call sum

pop result

; Вычтем y1\*y3

;fild word ptr [y1]

;fild word ptr [y3]

;fmulp st(1), st(0)

;fchs

;faddp st(1), st(0)

push y1

push y3

call mult

push result

call minus

pop result

; Сравним результат с 0

cmp result, 0

je pryamougolniy ; переход к точке pryamougolniy, если прямоугольный

; mov otvet, 0

mov result, 0

mov esp, ebx

ret ; если не прямоуг., то выход из функции

;fldz

;fcom st(1)

;fstsw ax

;sahf

pryamougolniy:

mov otvet, 0

mov esp, ebx

ret

vichislenia ENDP

;obhod:

; ; Треугольник прямоугольный

; invoke ExitProcess, otvet

;

exit:

; ; Треугольник не прямоугольный

invoke ExitProcess, otvet

;otvet = 0 --> является прямоугольным

;otvet = 1 --> неявляется прямоугольным

End Start

Вывод:

В данной работе я изучил принципы выполнения арифметических команд с помощью математического сопроцессора FPU микропроцессоров с архитектурой х86 в MASM.

* .686 – данная директива указывает ассемблеру использовать набор команд процессора Pentium Pro или Pentium II.
* .model flat - это несегментированная модель памяти, используемая в 32-разрядных операционных системах. В этой модели все данные и код программы размещаются в одном логическом адресном пространстве, что упрощает адресацию памяти. Сегментные регистры не используются для адресации данных, а вместо них используются 32-разрядные смещения
* .model stdcall - это соглашение о вызовах процедур, которое определяет порядок передачи параметров и очистки стека. При использовании stdcall, параметры передаются через стек в обратном порядке, а очистка стека производится вызываемой процедурой
* .stack100h – данная директива определяет размер стека программы. В данном случае, стек будет иметь размер 256 байт (100h — это шестнадцатеричное представление 256 в десятичной системе)
* ExitProcess PROTO STDCALL: DWORD:
  + ExitProcess:
    - Это имя функции из Windows API, которая завершает выполнение текущего процесса.
    - Она принимает один параметр — код завершения процесса (тип DWORD).
  + PROTO:
    - Ключевое слово MASM для объявления прототипа функции.
    - Прототип позволяет ассемблеру знать, как вызывать функцию, какие параметры она принимает и в каком порядке.
  + STDCALL:
    - Указывает соглашение о вызове функции.
    - В соглашении stdcall параметры передаются через стек в обратном порядке (сначала последний параметр), а очистка стека выполняется вызываемой функцией.
    - Это стандартное соглашение для большинства функций Windows API.
  + DWORD:
    - Описывает тип параметра функции. В данном случае функция принимает один параметр типа DWORD (32-битное беззнаковое целое число).
* API (Application Programming Interface) — это набор способов и правил, по которым различные программы общаются между собой и обмениваются данными.