Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Алтайский государственный технический университет

им. И.И. Ползунова»

Факультет (институт) Информационных технологий

Кафедра Прикладная математика

Отчет защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_

А.И.Потупчик

(подпись преподавателя) (инициалы, фамилия)

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Отчет

по лабораторной (практической) работе №\_6\_

Система команд процессора 80х86 в реальном режиме

(название лабораторной (практической) работы)

по дисциплине Архитектура ЭВМ

(наименование дисциплины)

ЛР 09.03.04.21.000 ОТ

(обозначение документа)

Студент группы ПИ-02 Р.А. Чередов

(инициалы, фамилия)

Преподаватель доцент, доцент А.И.Потупчик

(должность, ученое звание) (инициалы, фамилия)

Барнаул 2022

**Тема:**

Логический состав процессора компьютера и назначение его компонентов. Принципы программного управления

**Цель лабораторной работы:**

Данная лабораторная работа посвящена знакомству с архитектурой и системой команд процессоров семейства Intel80x86.

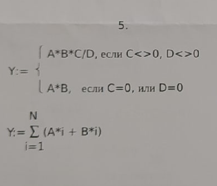
**Задание к лабораторной работе:**

* ознакомиться с базовой моделью программирования процессоров семейства Intel80x86, распределением адресного пространства, системой команд, методами адресации;
* ознакомиться с системой команд и использованием системного отладчика Debug (см. Приложение 4);
* для заданного варианта задания реализовать программы, используя систему команд семейства процессоров Intel80x86;
* используя отладчик, ввести программы в память компьютера и выполнить их в непрерывном и пошаговом режиме, наблюдая результаты выполнения команд по содержимому регистров процессора и оперативной памяти;
* оформить отчет по лабораторной работе.

**Задание:**

* Реализовать линейный, ветвящийся и циклический алгоритмы, используя систему команд процессора 80х86 реального режима.
* Программы реализуются в среде DosBox или FreeDos c использованием отладчика debug.
* Линейный алгоритм реализуется одним способом, ветвящийся алгоритм реализуется одним способом, циклический алгоритм реализуется тремя способами.

Y=a\*b+(c^2-d)



# Линейный алгоритм

Текст программы:

/\*

Y=a\*b+(c^2-d)

A – [200]

B – [202]

C – [204]

D – [206]

\*/

mov ax, [204] в регистр AX заносится содержимое слова DS:204 - C imul ax содержимое AX умножается на AX(c^2)

sub ax,[206] содержимое AX-d

mov bx,ax содержимое AX переносится в BX

mov ax,[200] в регистр AX заносится содержимое слова DS:200 - A

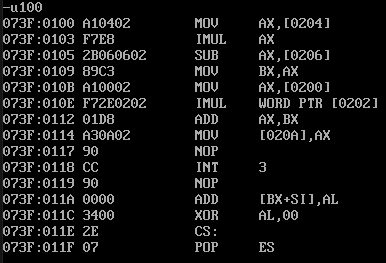
imul word ptr [202] содержимое AX\*B

add ax,bx к содержимому AX добавить BX (a\*b+(c^2-d))

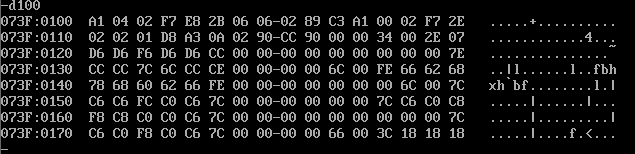
mov [20a],ax занести результат в ячейку c адресом DS:20a nop пустая команда

int 3 вернуться в отладчик nop пустая команда

**Команда дизассемблирования:**



**Расположение команды в памяти:**



**Тесты программы:**

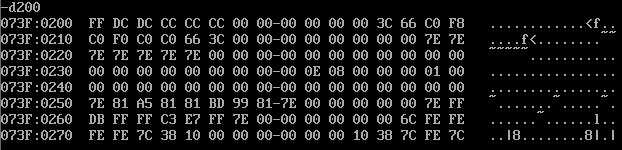
**Входные данные:**

**A = 2, B = 2, C = 2, D = 2**

**Выходные данные:**

**Y=2\*2+(4-2)=6**

Состояние ячеек памяти и регистров перед началом программы:



Результат работы:



# Ветвящийся алгоритм

Текст программы:

/\*

#### /A\*B\*C/D, если C<>0 и D<>0

**Y : = {**

#### \ A\*B, если C=0 или D=0

A – [200]

B – [202]

C – [204]

D – [206]

\*/

mov ax,[200]

imul word ptr [202]

mov word ptr[20a],ax

mov ax,[206]

imul ax,[204]

jz 011C

mov ax,word ptr[20a]

imul ax,[204]

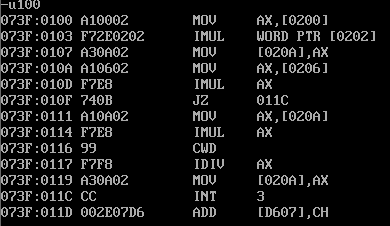
cwd

idiv ax,[206]

mov word ptr[20a],ax

int 3

**Команда дизассемблирования:**



**Тесты программы:**

**Входные данные:**

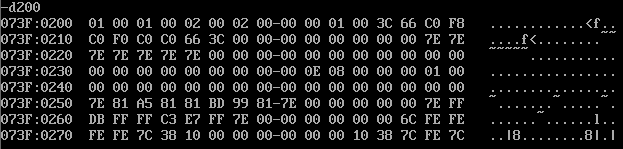
**A = 1, B = 1, C = 2, D = 2**

**Выходные данные:**

**С < > 0 и D < > 0**

**Y = A\*B\*C / D= 1\*1\*2/ 2 = 2 / 2 = 1**

Результат работы:



**Входные данные:**

**A = 1, B = 1, C = 0, D = 1**

**Выходные данные:**

**C=0 или D=0**

**Y = A\*B= 1\*1=1**

Состояние ячеек памяти и регистров перед началом программы:



Результат работы:



Цикл с предусловием

Текст программы:

/\*

N – [200]

A – [202]

B– [204]

\*/

mov cx,word ptr[200]

cmp cx,000

jz 012D

cmp cx,000

jz 0119

mov ax,cx

imul word ptr [202]

add bx,ax

dec cx

jmp 0109

mov cx,word ptr[200]

cmp cx,000

jz 012D

mov ax,cx

imul word ptr[204]

add bx,ax

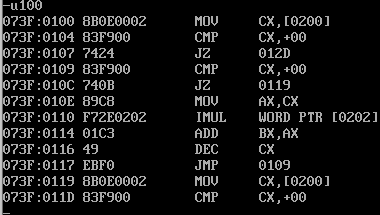
dec cx

jmp 011D

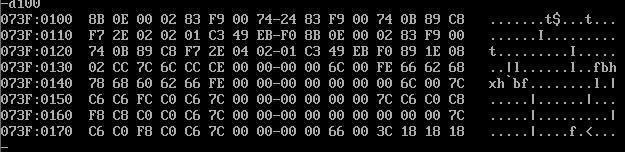
mov word ptr[208],bx

int 3

**Команда дизассемблирования:**



**Расположение команды в памяти:**



**Тесты программы:**

**Входные данные:**

**A = 2, B = 2, N = 2**

### Выходные данные:

**Y = 2\*2+2\*1+2\*2+2\*1=12**

Результат работы:



## Цикл с постусловием

Текст программы:

/\*

N – [200]

A – [202]

B – [204]

\*/

mov cx,word ptr[200]

mov ax,cx

imul word ptr[202]

add bx,ax

dec cx

cmp cx,0000

jnz 0104

mov cx,word ptr[200]

mov ax,cx

imul word ptr[204]

add bx,ax

dec cx

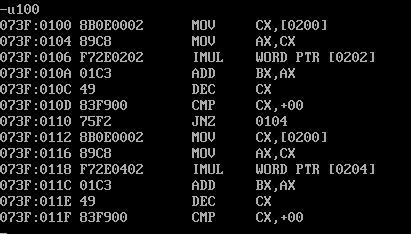
cmp cx,0000

jnz 0116

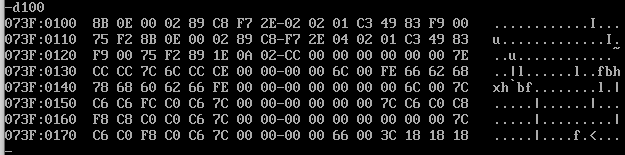
mov word ptr[20a],bx

int 3

**Команда дизассемблирования:**



**Расположение команды в памяти:**



**Тесты программы:**

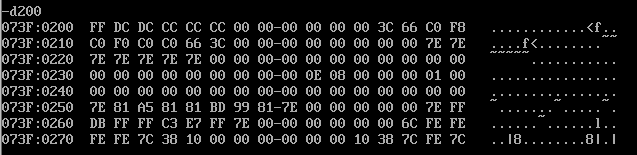
**Входные данные:**

**A = 2, B = 2, N = 2**

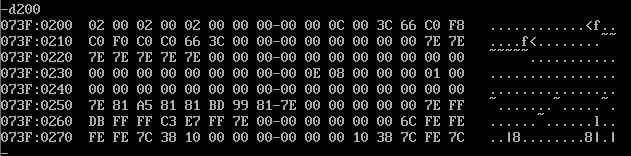
### Выходные данные:

**Y = 2\*2+2\*1+2\*2+2\*1=12**

остояние ячеек памяти и регистров перед началом программы:



Результат работы:



## Цикл с loop

Текст программы:

/\*

N – [200]

A – [202]

B – [204]

\*/

**mov cx,word ptr[200]**

**mov ax,cx**

**imul word ptr[202]**

**add bx,ax**

**mov ax,cx**

**imul word ptr[204]**

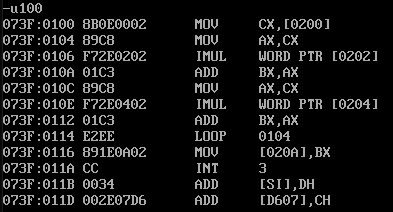
**add bx,ax**

**loop 0104**

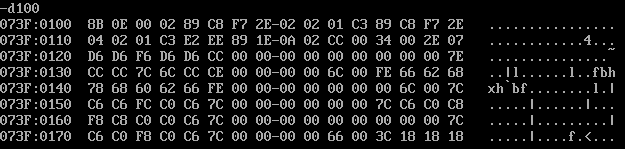
**mov word ptr[20a],bx**

**int 3**

**Команда дизассемблирования:**



**Расположение команды в памяти:**



**Тесты программы:**

**Входные данные:**

**A = 2, B = 2, N = 2**

### Выходные данные:

**Y=2\*2+2\*2+2\*1+2\*1=12**

Результат работы:

