**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕЬЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА(ЛЕНИНА)**

**Кафедра ВТ**

**ОТЧЕТ**

**По лабораторной работе №1**

**По дисциплине «Программирование»**

**По теме:**

**«Инструментальные средства разработки программ»**

Выполнил

Студент гр.9892 Евсеева В. Д.

Преподаватель Валов А.А.

Санкт-Петербург

2021

**Цель: освоение средств отладки программ.**

Список заданий:

1. Выполнение программы вычисления N чисел Фибоначчи.
2. Осуществить инкрементирование регистра AX пять раз.
3. Осуществить инкрементирование ячейки с адресом 5 пять раз.
4. Осуществить пять раз инкрементирование ячейки с адресом 5, используя косвенную адресацию.
5. Переместить содержимое однобайтовых ячеек, имеющих смещение [00] … [07], в ячейки со смещением [08] … [0F]. Ячейки размещены в сегменте, начальный адрес которого задан в регистре DS.

**Задание 1**

Текст программы:

MOV AX,0; AX = 0

MOV BX,1; BX = 1

MOV DX,0; DX = 0

MOV CX,N; поместить в конкретное число

M1: ADD AX,BX; в AX последовательно формируются числа Фибоначчи

MOV BX,DX

MOV DX,AX

LOOP M1; цикл, М1 – адрес метки, на которую возвращается LOOP, пока СХ не равно 0

Команда (инструкция) mov копирует значение из приемника в источник (mov приемник, источник)

Команда (инструкция) loop переходит по метке, уменьшает значение регистра cx на единицу, проверяет равно ли cx нулю.

Команда (инструкция) add складывает два (целочисленных) операнда источник и приемник размерностью байт, слово или двойное слово.

Для выполнения данного задания, я взяла N = 7.

В моем случае, М1 = 010С.

Программа вычисляет N чисел Фибоначчи (Числа Фибоначчи — это ряд, состоящий из целых чисел. Их особенность заключается в том, что каждый элемент представляет собой сумму двух предыдущих чисел).

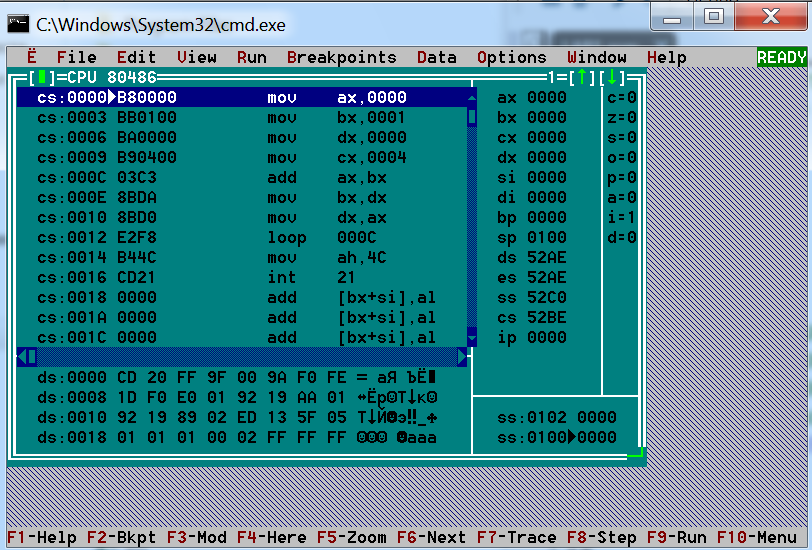


Рис. 1.1 – Программа до отладки

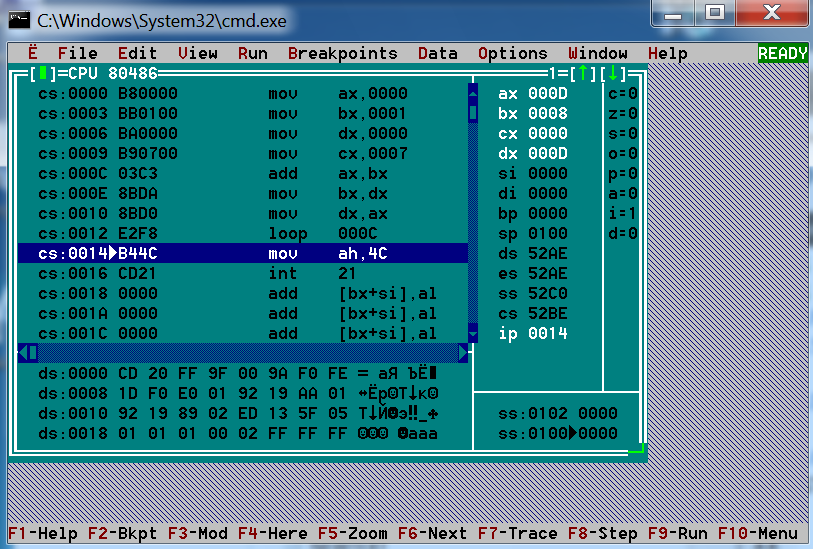


Рис. 1.2 – Программа после отладки

Результат программы – ax = 000D в шестнадцатеричной системе, что является 13 в десятеричной системе счисления.

**Задание 2**

Текст программы:

Mov ax,0; ax = 0000

Mov cx,7; cx = 0007

Inc ax; ax = ax + 1

Loop 0106; цикл, 0106 – адрес метки, на которую возвращается loop, пока cx не равно 0

Команда (инструкция) inc выполняет инкрементирование переменной, то есть увеличение на единицу.

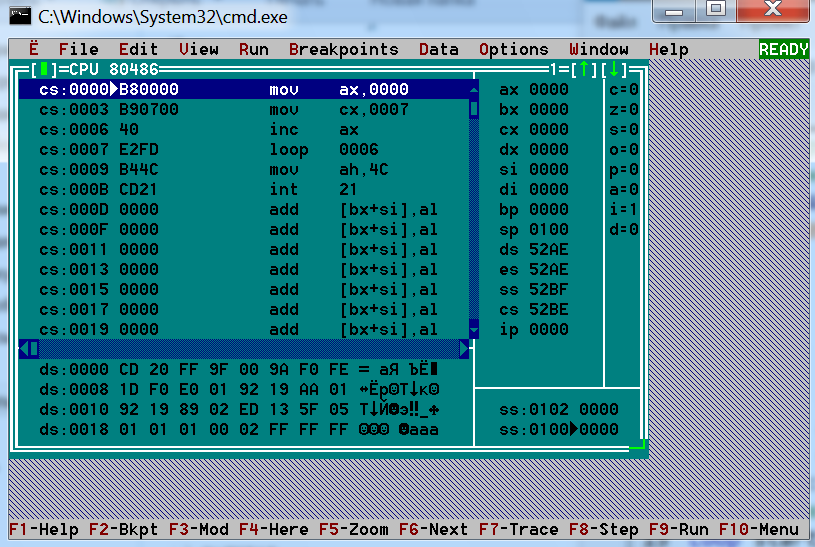


Рис. 2.1 – Программа до отладки

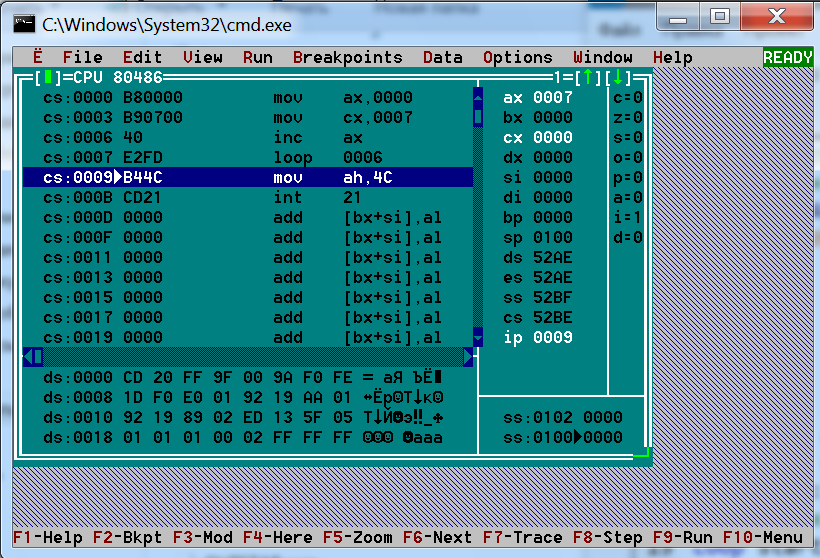


Рис. 2.2 – Программа после отладки

Результат работы программы:

Значение регистра ax увеличилось на единицу 7 раз (то есть увеличилось на 7; ax = 0007), регистр cx уменьшился на 7, так как инструкция loop уменьшает cx на единицу при каждом прохождении цикла и прекращает работу, когда cx = 0000.

**Задание 3**

Текст программы:

Mov [0005],1; в ячейку [0005] записываем значение 1

Mov cx,6; загружаем в сх количество итераций

Inc byte ptr [0005]; инкремент ячейки с косвенной адресацией

Loop 0108; цикл, 0108 – адрес метки, на которую возвращается loop, пока cx не равно 0

В данной программе используется непосредственная адресация. Прямой операнд имеет константное значение или выражение. Когда инструкция с двумя операндами использует прямую адресацию, то первый операнд может быть регистром или ячейкой памяти, а второй — непосредственной константой.

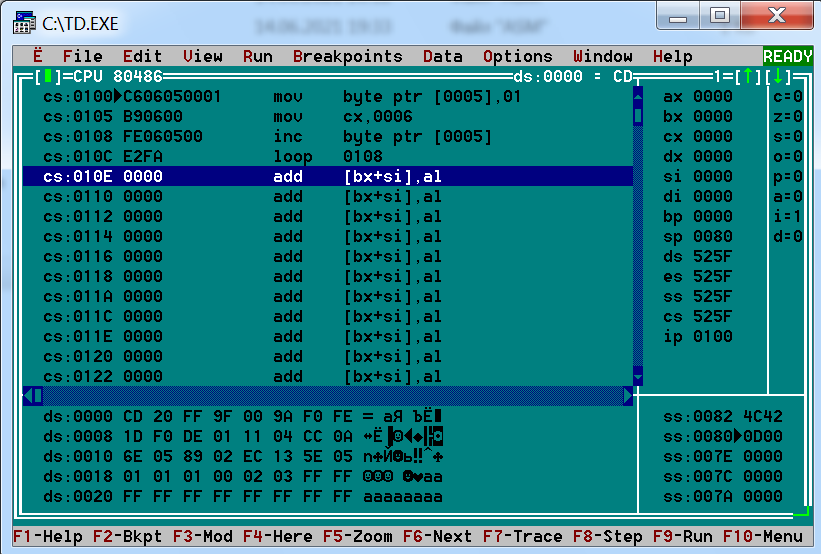


Рис. 4.1 – Программа до отладки

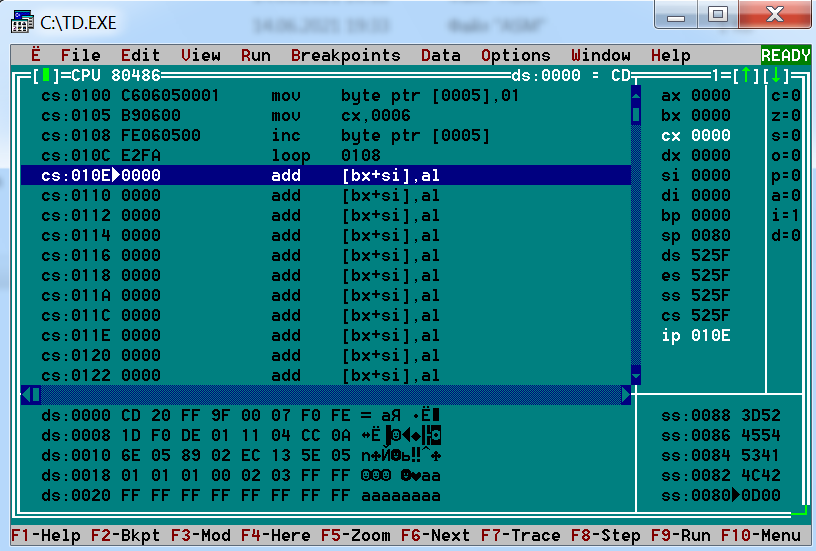


Рис. 4.2 – Программа после отладки

Результат программы:

Работаем с ячейкой [0005]. В результате ее значение увеличивается на единицу 6 раз.

**Задание 4**

Текст программы:

Прежде чем прогонять программу, записываем в ячейку [0005] любое значение, в нашем случае пусть это будет [0005] = 2.

Mov ax,[0005]; записываем в регистр ах содержимое ячейки с адресом 0005

Mov cx,0006; cx = 0006

Inc ax; ax = ax + 1

Loop 0106; цикл, 0106- адрес метки, на которую возвращается loop, пока cx не равно 0

Mov [0005],ax; записываем в ячейку с адресом 0005 содержимое регистра ах при помощи прямой адресации

В программе используется косвенная и прямая адресация. При такой адресации эффективный адрес операнда может находиться в любом из регистров общего назначения, кроме sp/esp и bp/ebp (это специфические регистры для работы с сегментом стека). Синтаксически в команде этот режим адресации выражается заключением имени регистра в квадратные скобки [ ].

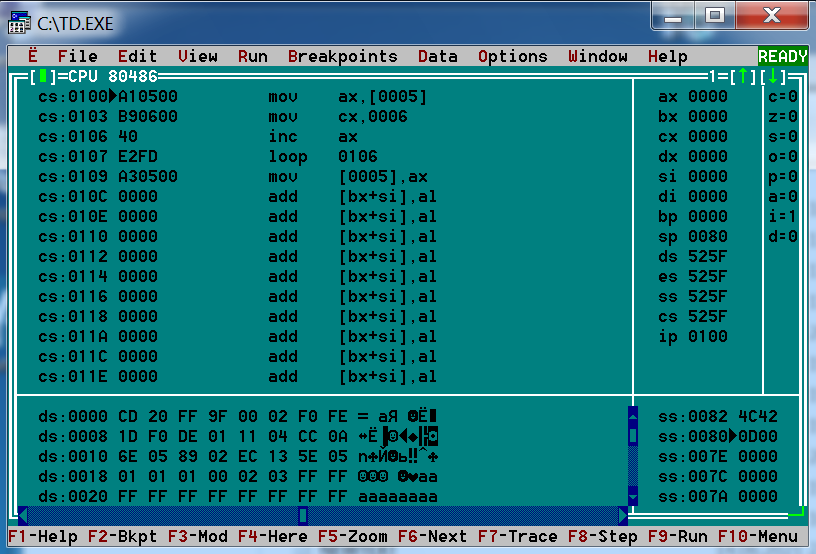


Рис. 3.1 – Программа до отладки

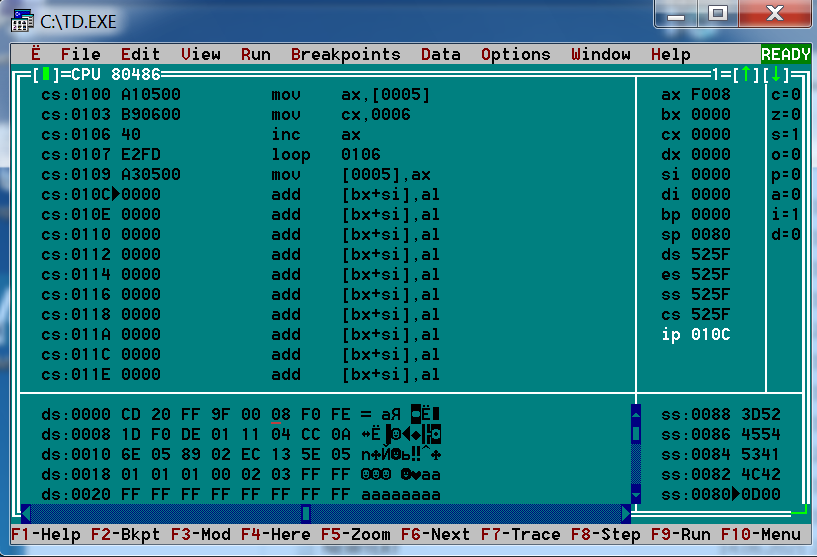


Рис. 3.2 – Программа после отладки

Результат работы программы:

Записываем данные из ячейки [0005] в регистр (косвенная адресация). Выполняем увеличение регистра на один 6 раз, затем полученное значение передается в ячейку с адресом 5 (прямая адресация). В итоге, в ячейку [0005] записывается значение из регистра ax, оно равно 08.

**Задание 5**

Текст программы:

Mov ax,0000; ax = 0000

Mov cx,0008; cx = 0008

Inc ax; ax = ax + 1

Mov [bx + si],ax; [bx + si] = ax

Inc si; si = si + 1

Loop 0106;

Mov cx,0008; cx = 0008

Sub si,0008; si = si - 8

Mov ax,[bx + si]; ax = [bx + si]

Add si,0008; si = si + 8

Mov [bx + si],ax; [bx + si] = ax

Inc si; si = si + 1

Loop 010F;

В данной программе используются два цикла. Первый для заполнения ячеек со смещением [00]…[07], второй для заполнения ячеек смещением [08]…[0F] значениями ячеек из предыдущего цикла.

Этот вид адресации является дополнением предыдущего и предназначен для доступа к данным с известным смещением относительно некоторого базового адреса. Этот вид адресации удобно использовать для доступа к элементам структур данных, когда смещение элементов известно заранее.

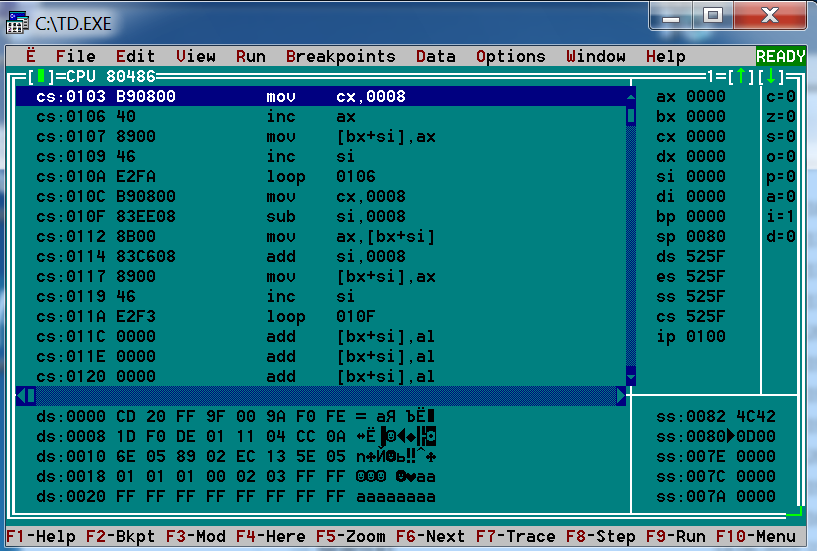


Рис. 5.1 – Программа до отладки

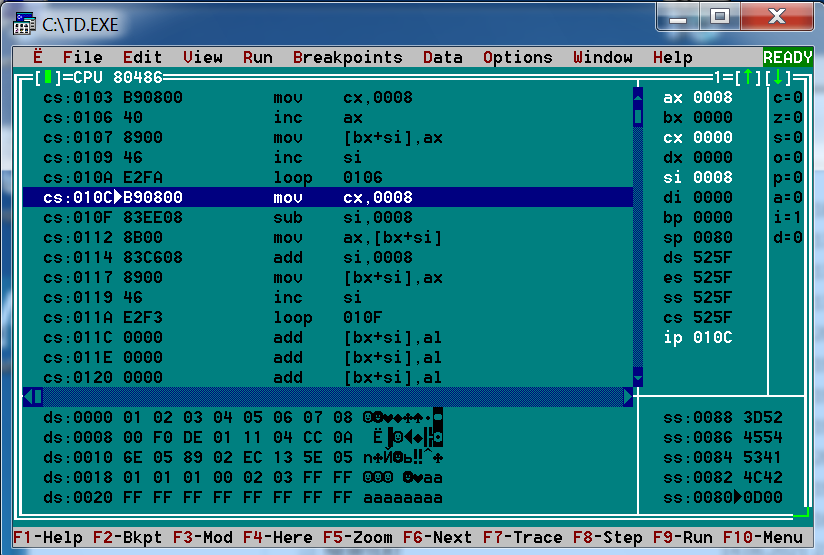


Рис. 5.2 Заполнение ячеек [00]…[07]

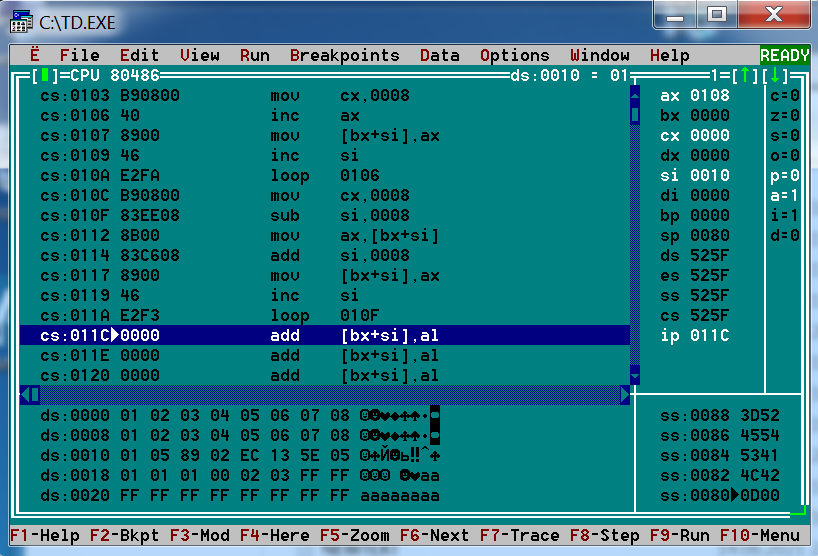


Рис. 5.2 – Программа после отладки

Результат программы:

Ячейки со смещением [08]…[0F], были заполнены данными из ячеек со смещением [00]…[07]

**Вывод:**

В ходе лабораторной работы были освоены средства отладки программ, способы адресации (непосредственная, косвенная, со смещением). Изучены команды: mov, inc, add, loop.

Получены следующие навыки:

* Структурированное решение задач
* Использование операторов языка
* Формирование тестовых ситуаций
* Осуществление процесса отладки
* Документирование результатов работы
* Использование Turbo Debugger