**Цель:** **освоение приемов программирования**

Список заданий.

1. Требуется написать и отладить программу на языке ассемблера, эквивалентную следующему фрагменту программы на языке C.

If((x > y) && (z < t) || (a != b)) c = d.

1. Требуется написать и отладить программу, реализующую структуру управления CASE, используя таблицу переходов

**Задание 1**

Текст программы:

TITLE LAB4

.Model Small

.STACK 100h

.DATA

a DB 0,1,2,3,4,5,6,7

b DB 8 dup(?)

auth DB 'VIKTOR','$'

mess DB 'NAZMITE KNOPKU DLY VIHODA $'

.CODE

START:

mov ax,@data

mov ds,ax

mov cx,8

lea si,a

lea di,b

label1:

mov al,[si]

mov [di],al

inc si

inc di

loop label1

mov ah,a+4 ;переменная x

mov al,a+5 ;переменная y

cmp ah,al

ja cmp\_z\_and\_t

jmp a\_not\_equal\_b

cmp\_z\_and\_t:

mov ah,a+6 ;переменная z

mov al,a+7 ;переменная t

cmp ah,al

jb move\_d\_to\_c

a\_not\_equal\_b:

mov ah,a ;переменная a

mov al,a+1 ;переменная b

cmp ah,al

je stop

move\_d\_to\_c:

mov al,a+3 ;переменная d

mov b+2,al ;переменная c=d

stop:

mov dl,b+2

or dl,30h

mov ah,02h

int 21h

mov ah,09h

lea dx,auth

int 21h

lea dx,mess

int 21h

mov ah,1

int 21h

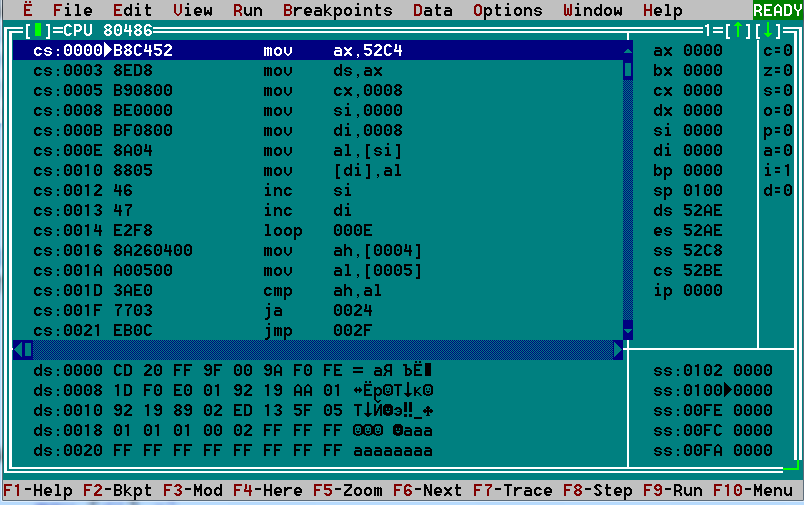
mov ax,4c00h

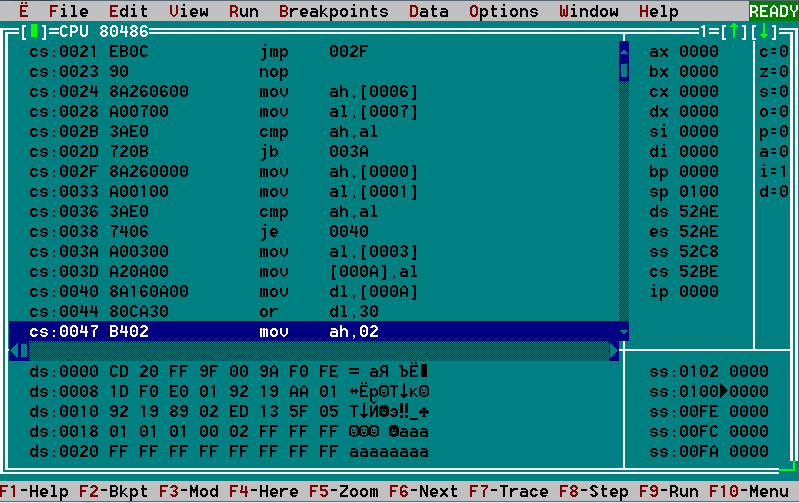
int 21h

END START

В программе сначала проверяется условие x > y (с помощью команд *cmp и ja,* которые поверяют, что первая переменная больше второй*)* , если оно выполняется, то по метке *m1* проверяется следующее условие z < t (с помощью команд *cmp и jb,* которые проверяют, что первая переменная меньше второй), если выполняются оба условия, то выполнить переход на метку m2 и присвоение *c=d.* Если не выполняется первое условие, то происходит переход на метку *anotb,* в которой проверяется последнее условие a != b. В программе это условие проверяется наоборот, то есть происходит проверка равенства этих переменных (a = b) с помощью команд *cmp и je.* Если выполняется равенство, то происходит переход на метку m3 и все переменные остаются без изменения, то есть c!=d.

Иллюстрация работы программы:





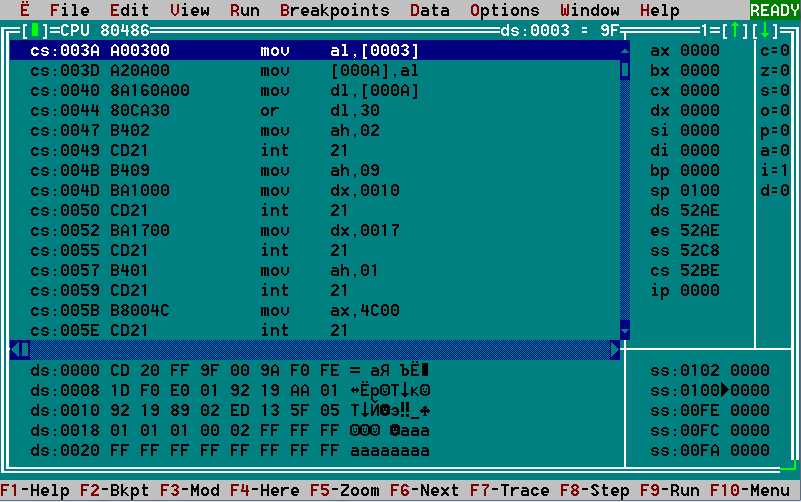


Рис. 1.1 – 1.3 – Программа до отладки

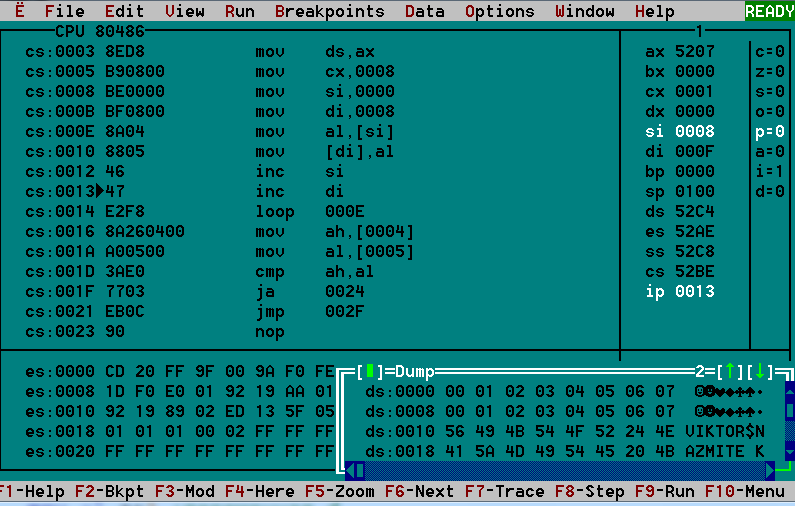


Рис. 1.4– Заполнение элементов массивов.

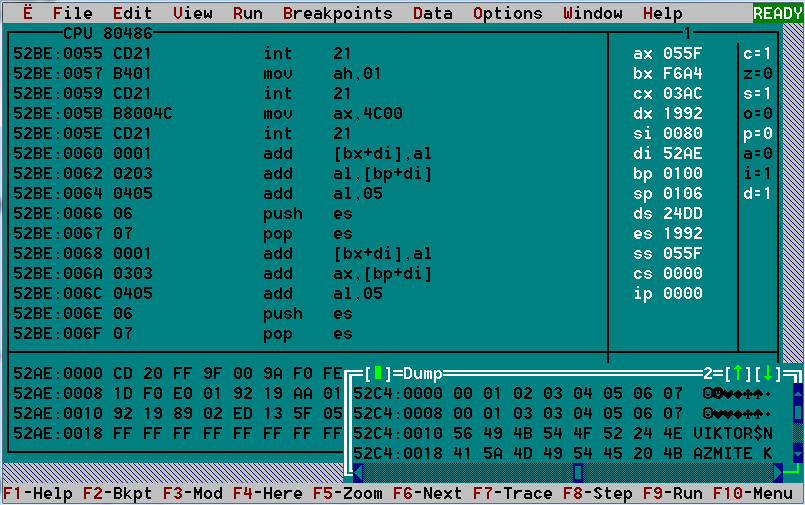


Рис. 1.5 – Программа после отладки

В итоге c = d = 03, так как выражение истинно.

Покажем работу программы еще на нескольких примерах.

a DB 0,1,2,3,5,4,6,7

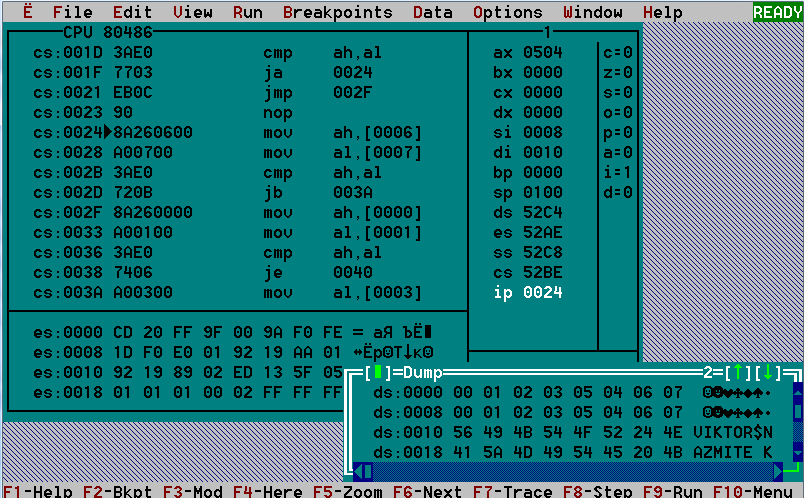
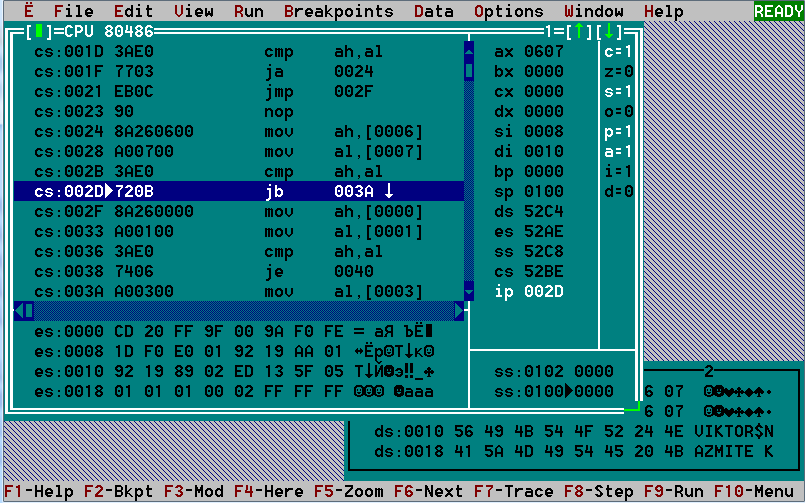


Рис. 1.6 – В ходе выполнения программы x > y = true, поэтому выполняется код cmp\_z\_and\_t



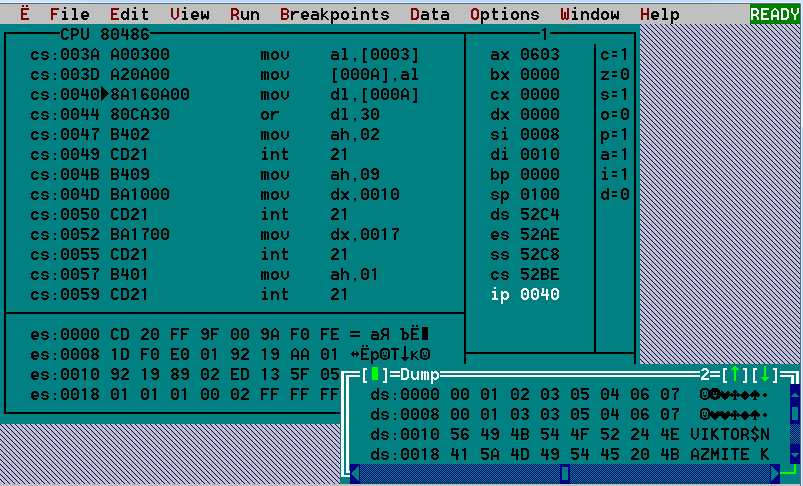


Рис. 1.7-1.8 Т.к. z < t = true, то сразу исполняется код c=d

a DB 0,0,2,3,5,4,7,7

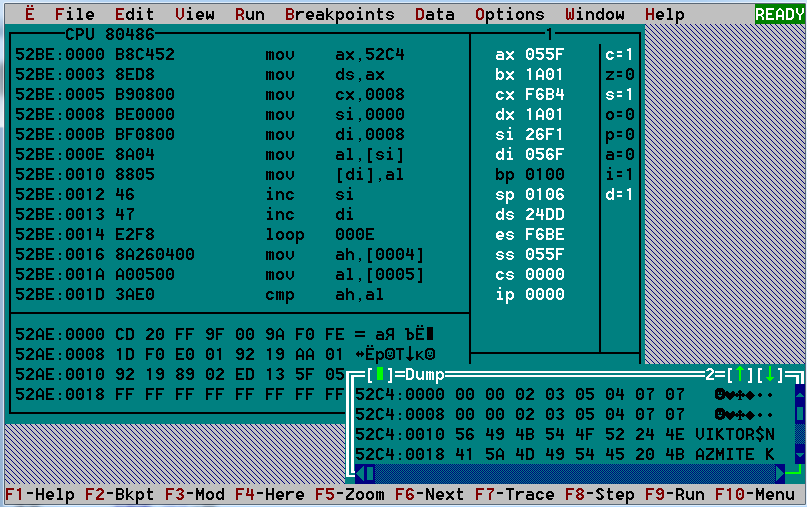


Рис. 1.9 – Программа после отладки

В итоге c != d, так как выражение ложно.

**Задание 2**

Требуется написать и отладить программу, реализующую структуру управления CASE, используя таблицу переходов

Текст программы:

TITLE LAB4

.Model Small

.STACK 100h

.DATA

jump\_table dw foo0, foo1, foo2; таблица переходов

a DB 0,1,2,3,4,5,6,7

b DB 8 dup(?)

.CODE

START:

mov ax,@data

mov ds,ax

mov bx,2; значение от 0 до 2 определяет набор операторов, соответствующих метке выбора

shl bx,1; умножение на два (если размер адреса в таблице переходов составляет 4 байта)

Jmp cs:jump\_table[bx]

foo0: ; 1 сравнение на 0, если верно то переход на метку case0

call case0

jmp exit

foo1:

call case1

jmp exit

foo2:

call case2

jmp exit

case0:

lea si, a; записываем эффективный адрес исходных данных в регистр si

lea di, b; записываем эффективный адрес исходных данных в регистр di

mov cx,8; счетчик для команды loop

loop1:

mov al, [si]; записываем в младшую часть регистра ax данные, на которые указывает адрес в регистре si

mov [di], al; записываем данные из al в ячейку результирующей последовательности, на которую ссылается адрес регистра di

inc si

inc di

loop loop1; повторяем действия в метке loop1, пока cx != 0

ret; пустая последовательность операторов

case1:

lea si, a; записываем эффективный адрес исходных данных в регистр si

lea di, b+7; устанавливается смещение для участка памяти, выделенную под запись, чтобы данные копировались с конца в обратном порядке

mov cx,8; счетчик для команды loop

loop2:

mov al, [si]; записываем в младшую часть регистра ax данные, на которые указывает адрес в регистре si

mov [di], al; записываем данные из al в ячейку результирующей последовательности, на которую ссылается адрес регистра di

inc si

dec di

loop loop2; повторяем действия в метке loop2, пока cx != 0

ret; пустая последовательность операторов

case2:

mov cx,8; счетчик для команды loop

lea si, a+7;

lea di, b+7; устанавливается смещение для участка памяти, выделенную под запись, чтобы данные копировались с конца в обратном порядке

loop3:

mov al, [si]; записываем в младшую часть регистра ax данные, на которые указывает адрес в регистре si

cmp al,5; сравнение значения в байте со значением 5

jg higher\_than\_5; если значение строго больше 5, то перепрыгиваем на метку higher\_than\_5, для прохождения цикла

mov al,0

higher\_than\_5:

mov [di],al

dec si

dec di

loop loop3; повторяем действия в метке loop1, пока cx != 0

ret; пустая последовательность операторов

exit:

mov ax,4C00h

int 21h

END START

В программе реализуется структура управления CASE с использованием таблицы переходов.

**Иллюстрация работы программы:**

Рис. 2.1 – Программа после отладки.

В итоге:

bx=0

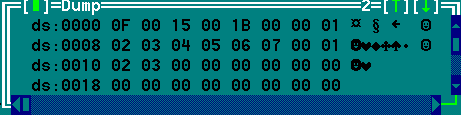


Рис. 2.2 Промежуточный результат заполнения массива b последовательно

bx=1

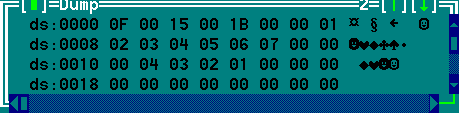


Рис. 2.3 Промежуточный результат заполнения массива b в обратном порядке

bx=2

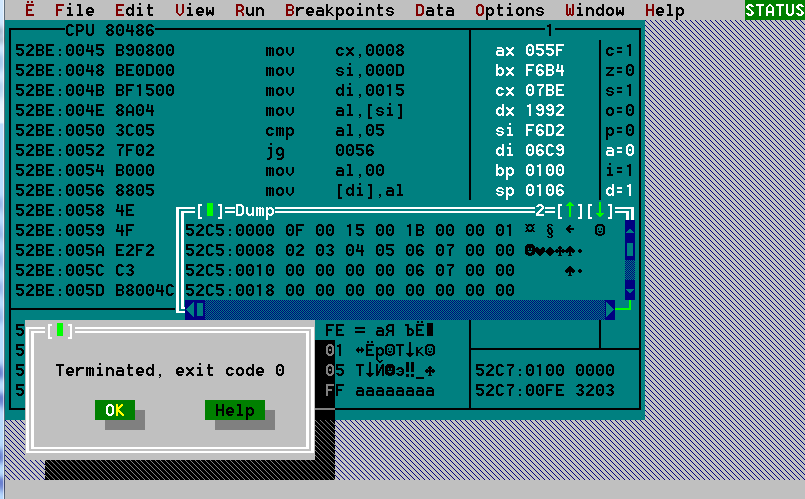


Рис. 2.4 Результат заполнения массива b с условием.

**Вывод:**

В результате данной лабораторной работы были освоены управляющие структуры и получен опыт работы с таблицами переходов CASE.

Получены следующие навыки:

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