МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий и управления в технических системах

кафедра «Информационные системы»

Лабораторная работа №4

«Исследование методов программирования обработки строковых данных»

по дисциплине «Технические средства информационных систем»

Выполнил: ст. гр. ИС/б-20-2-о

Белик Г. М.

Проверил: Минкин С. И.

Севастополь

2022 г.

Цель работы:

Изучить основные команды языка ассемблера для обработки строковых данных и команды передачи управления, исследовать их воздействие на процесс ассемблирования и формирования листинга программы.

Исследовать особенности функционирования блоков 16–разрядного микропроцессора при выполнении команд обработки строк и передачи управления. Приобрести практические навыки программирования на языке ассемблера МП 8086 задач обработки линейных массивов.

Постановказадачи:

1. Повторить основные директивы ассемблера и их воздействие на процесс ассемблирования и формирования листинга программы. Изучить команды строковых операций (выполняется в процессе домашней подготовки к лабораторной работе).

2. Изучить команды передачи управления в 16–разрядных процессорах и особенности оформления программ в ехе– и сом–форматах (выполняется во время домашней подготовке к работе).

3. Составит программу, состоящую из следующих процедур обработки строк:

3.1. Исследовать программу пересылки массивов, приведенную в приложении. Оптимизировать данную программу за счет использования префикса повторения.

3.2. Заполнить 100+10i ячеек области памяти, начинающейся с адреса MAS1 рядом натуральных чисел. Здесь i – последняя цифра номера Вашей зачетной книжки.

3.3. Переслать массив слов из области памяти, начиная с адреса MAS1 в область с начальным адресом MAS2.

3.4. Найти в заданном массиве число, равное двум последним цифрам Вашей зачетной книжки и определить его индекс.

3.5. Вычислить сумму элементов массива MAS1, созданного в п. 2.3.1.

4. Произвести отладку разработанных программ в пошаговом режиме и проследить за изменениями содержимого регистров.

Ход работы

Программа из листинга 1 была оптимизирована за счет префикса повторения. Результат всех преобразований программы из задания представлен в листинге 2.

Листинг программы 1 – Начальная программа

; multi–segment executable file template.

data segment

NAME1 DB 'ABCDEFGHI'

NAME2 DB 'JKLMNOPQR'

NAME3 DB 'STUVWXYZ\*'

ends

stack segment

dw 128 dup(0)

ends

code segment

start:

; set segment registers:

mov ax, data

mov ds, ax

mov es, ax

LEA SI,NAME1 ; address initialization

LEA DI,NAME2 ; NAME1 and NAME2

MOV CX,09 ; Forward 9 characters

B20:

MOV AL,[SI] ; Forward from NAME1

MOV [DI],AL ; Forward in NAME2

INC SI ; Next symbol in NAME1

INC DI ; Next position in NAME2

DEC CX ; Decrease cycle counter

JNZ B20 ;Counter > 0? Yes - loop

;RET ;Counter = 0, then break the loop

LEA SI,NAME2 ; address initialization

LEA DI,NAME3 ; NAME2 and NAME3

MOV CX,09 ;Forward 9 symbols

C20:

MOV AL,[SI] ; Forward from NAME2

MOV [DI],AL ; Forward in NAME3

INC DI ;Next symbol in NAME2

INC SI ; Next position in NAME3

LOOP C20 ;Decrease counter,

; <>0 --> loop

; =0 --> breakя

mov ax, 4c00h ; exit to operating system.

int 21h

ends

end start ; set entry point and stop the assembler.

Листинг 2 – Итоговая программа

; multi–segment executable file template.

data segment

NAME1 DB 'ABCDEFGHI'

NAME2 DB 'JKLMNOPQR'

NAME3 DB 'STUVWXYZ\*'

array db 100

ends

stack segment

dw 128 dup(0)

ends

code segment

start:

; set segment registers:

mov ax, data

mov ds, ax

mov es, ax

;--------------------1------------------

CLD

MOV CX,09

LEA DI,NAME2

LEA SI,NAME1

REP MOVSB

;---------------------------------------

CLD ;

MOV CX,09 ;

LEA DI,NAME3 ;

LEA SI,NAME2 ;

REP MOVSB

;-------------------2-------------------

LEA DI,array

LEA SI,NAME1

mov AX, 1

mov CX,150

fill:

mov array[SI],AL

inc AX

add SI,1

loop fill

;-------------------3-------------------

CLD

MOV CX,09

LEA DI, NAME2

LEA SI, NAME1

REP MOVSW ;

;-------------------4--------------------

MOV CX, 150

xor ax, ax

xor si,si

mov si, 0

cycle:

inc si

mov aL, array[si]

cmp aL, 50

loopnz cycle

jz yes

yes:

mov ah, 09h

lea dx, [si]

int 21h

;-------------------5-------------------

mov cx, 9

mov ax, 0

lea si, name1

sum:

add al,name1[si]

inc si

loop sum

mov ax, 4c00h ; exit to operating system.

int 21h

ends

end start ; set entry point and stop the assembler.

Пояснения к коду.

Пункт 1. Требовалось исследовать программу пересылки массивов, приведенную в листинге 1 и оптимизировать данную программу за счет использования префикса повторения.

CLD ; сброс флага D

MOV CX,09 ; переместить в регистр СХ количество повторений цикла

LEA DI,NAME2 ; загрузить эффект. адрес области «куда»

LEA SI,NAME1 ; загрузить эффект. адрес области «откуда»

REP MOVSB ; повторять пересылку данных, пока СХ>0

;------------------------------------------------------------------

CLD ; сброс флага D

MOV CX,09 ; переместить в регистр СХ количество повторений цикла

LEA DI,NAME3 ; загрузить эффект. адрес области «куда»

LEA SI,NAME2 ; загрузить эффект. адрес области «откуда»

REP MOVSB ; повторять пересылку данных, пока СХ>0

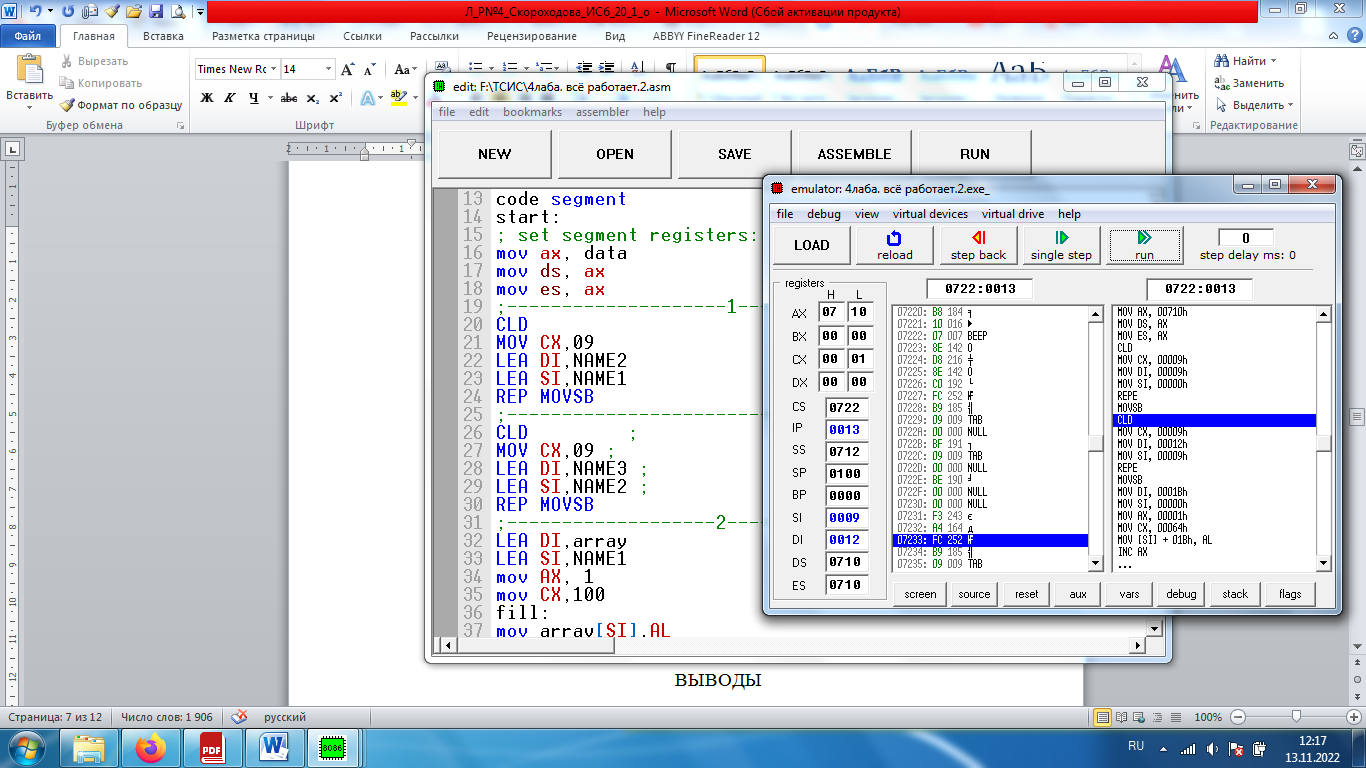


Рисунок 1 – Значение регистров SI, DI после одного цикла пересылки

Как видно, пересылка данных произошла успешно, поскольку смещение изменилось на 9 – на длину массива.

Пункт 2. Требовалось заполнить 100+10i ячеек области памяти, начинающейся с адреса MAS1 рядом натуральных чисел.

LEA DI,array ; загрузить эффект. адрес области «куда»

LEA SI,NAME1 ; загрузить эффект. адрес области «откуда»

mov AX, 1 ; поместить в регистр АХ 1 (первое натуральное число)

mov CX,150 ; поместить в регистр СХ число повторений (100, так

; как это длина массива натуральных чисел)

fill: ; цикл заполнения ячеек памяти

mov array[SI],AL ; в массив поместить число из регистра AL

inc AX ; инкрементировать число из регистра

add SI,1 ; инкрементировать регистр SI

loop fill

Пункт 3. Переслать массив слов из области памяти, начиная с адреса MAS1 в область с начальным адресом MAS2.

CLD ; сброс флага D

MOV CX, 09 ; переместить в регистр СХ количество повторений цикла

LEA DI, NAME2 ; загрузить эффект. адрес области «куда»

LEA SI, NAME1 ; загрузить эффект. адрес области «откуда»

REP MOVSW ; переслать массив слов

Пункт 4. Найти в заданном массиве число, равное двум последним цифрам Вашей зачетной книжки и определить его индекс.

MOV CX, 150 ; переместить в регистр СХ количество повторений цикла

; (150, потому что это длина массива)

xor ax, ax ; обнуление регистров

xor si,si ;

mov si, 0 ;

cycle: ; цикл поиска числа 50

inc si ; инкремент регистра SI

mov aL, array[si] ; поместить в регистр AL элемент массива

cmp aL, 50 ; сравнение содержимого регистра и числа 50

loopnz cycle ; если флаг ZF=1 🡪 выход из цикла

jz yes ;

yes: ;

mov ah, 09h ; прерывание, вывод на экран

lea dx, [si] ; в регистре DX – индекс искомого числа

int 21h

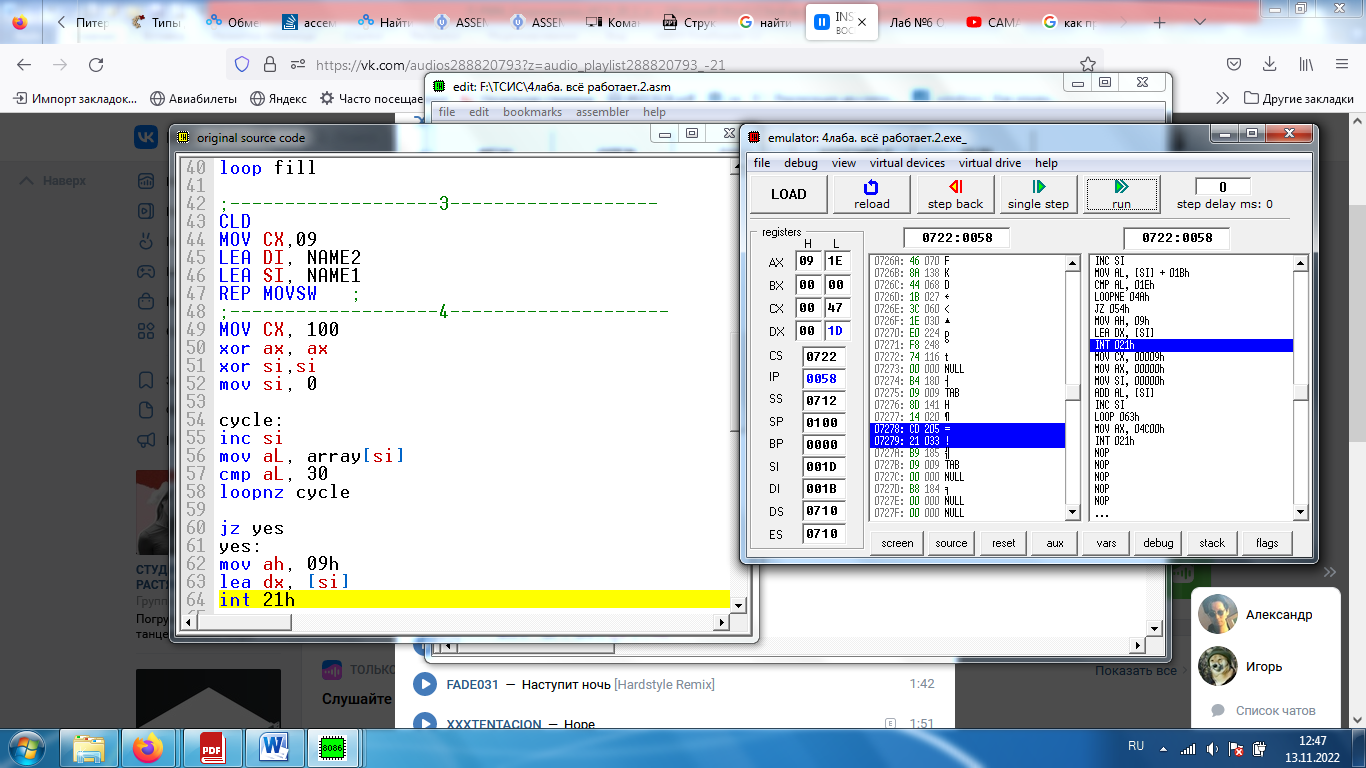


Рисунок 2 – Содержимое регистров после поиска числа

Пункт 5. Вычислить сумму элементов массива MAS1, созданного в п. 2.3.1.

Так как массив состоит из букв, складывались коды этих букв. Начиная с буквы А (код 41), коды увеличивались в зависимости от буквы.

mov cx, 9 ; переместить в регистр СХ число повторений цикла

mov ax, 0 ; обнулить регистр АХ

lea si, name1 ; загрузить эффект. адрес области «откуда»

sum: ; цикл нахождения суммы

add al,name1[si] ; добавить в регистр AL код элемента массива

inc si ; инкрементировать регистр SI

loop sum

ВЫВОДЫ

В ходе лабораторной работы были изучены основные команды языка ассемблера для обработки строковых данных и команды передачи управления, исследовано их воздействие на процесс ассемблирования и формирования листинга программы.

Была модифицирована программа в соответствии с заданием, написаны циклы обработки строк, такие как поиск искомого числа, вычисление суммы элементов массива.

**Контрольные вопросы:**

1. Команды порождают одну машинную команду. Директивы (псевдокоманды) содержат управляющую информацию для ассемблера. Директивы используются для распределения памяти, связей между модулями, манипуляции с символами и т.д.

2. Для определения и инициализации данных предназначены директивы:

DB - определить байт (Define Byte)

DW - определить слово (Define Word)

DD - определить двойное слово (Define Double Word).

Формат директивы: имя DX , [< начальное значение>]

Пример определения переменных:

Z1 DB 0ABH ; один байт, равный

AB Z2 DW 1000H ; одно слово, равное 1000

Z3 DD 1235H; 1000H ; младшее слово 1000, старшее 1235.

3. Для организации процедур в языке ассемблера предназначены директивы PROC и ENDP. Директива PROC отмечает точку входа процедуры, а директива ENDP –окончание процедуры.

Формат этих директив имеет вид:

[имя] PROC [тип]

. тело процедуры

[имя] ENDP

Имя представляет точку входа в процедуру. Тип процедуры (NEAR или FAR) (по умолчанию NEAR) используется ассемблером для определения вида генерируемой команды CALL для вызова процедуры.

4. Каждая программа содержит 3 типа сегментов:

Сегмент кодов – содержит машинные команды для выполнения. Обычно первая выполняемая команда находится в начале этого сегмента, и операционная система передает управление по адресу данного сегмента для выполнения программы. Регистр сегмента кодов (CS) адресует данный сегмент.

Сегмент данных – содержит данные, константы и рабочие области, необходимые программе. Регистр сегмента данных (DS) адресует данный сегмент.

Сегмент стека — содержит адреса возврата как для программы (для возврата в операционную систему), так и для вызовов подпрограмм (для возврата в главную программу), а также используется для передачи параметров в процедуры. Регистр сегмента стека (SS) адресует данный сегмент. Адрес текущей вершины стека задается регистрами SS:ESP.

5. Параметры директивы SEGMENT.

1) Выравнивание. Определяет границу начала сегмента. Обычное значение - PARA, по которому сегмент устанавливается на границу параграфа. В этом случае адрес кратен 16. При отсутствии этого операнда ассемблер по умолчанию принимает PARA; адрес сегмента ХХХ0. Бывает: PAGE=ХХ00; WORD=ХХХЕ (четная граница); BYTE=ХХХХ – любая шестнадцатеричная цифра.

2) Объединение. Определяет, объединяется ли данный сегмент с другими

сегментами в процессе компоновки после ассемблирования.

3) Класс. Данный элемент, заключенный в апострофы, используется для группирования относительных сегментов при компоновке. Он может содержать любое имя и используется ассемблером для обработки сегментов, имеющих одинаковые имена и классы.

6. Текущее значение счетчика адресов может быть принудительно изменено программистом с помощью директивы ORG <выражение>. При выполнении директивы ORG вычисляется значение выражения и полученный результат загружается в счетчик ячеек (адресов).

7) Директива ENDS с тем же именем отключает данный счетчик до тех пор, пока этот же сегмент не будет открыт еще одной директивой SEGMENT. В этом случае счетчик продолжает счет распределяемых байтов со старого значения.

8) Строкой (String) в языке ассемблера называют набор (цепочку) однотипных символов, байтов или слов. Она обычно заканчивается нулевым символом, который равен « 0». Строковые (цепочечные) команды используются для эффективной обработки массивов. Для обработки строковых данных ассемблер имеет пять команд обработки строк: MOVS – переслать один байт или одно слово из одной области памяти в другую; LODS – загрузить из памяти один байт в регистр AL или одно слово в регистр AX; STOS – записать содержимое регистра AL или AX в память; CMPS – сравнить содержимое двух областей памяти, размером в один байт или в одно слово; SCAS – сравнить содержимое регистра AL или AX с содержимым памяти.

9) DI Адрес области "куда" переслать.

SI Адрес области "откуда" переслать.

10) Для задания направления перемещения по строкам применяется флаг D, который устанавливается командой STD и сбрасывается командой CLD. При нулевом значении флага при каждом повторении происходит увеличение адресов источника и приемника, а при единичном – уменьшение.

11) Переслать Байт MOVSB.

Переслать слово MOVSW.

12) Для использования префикса REP необходимо установить количество повторений в регистре CX. При выполнении цепочечной команды с префиксом REP происходит уменьшение на 1 значения в регистре CX до нуля. Таким образом, можно обрабатывать строки любой длины.

13) REP – повторять операцию, пока CX не равно 0;

REPZ или REPE – повторять операцию, пока флаг ZF показывает "равно или ноль". Прекратить операцию при флаге ZF, указывающему на не равно или не ноль или при CX равном 0;

REPNE или REPNZ – повторять операцию, пока флаг ZF показывает "не равно или не ноль". Прекратить операцию при флаге ZF, указывающему на "равно или нуль" или при CX равным 0.

14) Передача управления в пределах текущего сегмента кода называется внутрисегментной – при этом модифицируется только регистр IP и адрес перехода может быть представлен, одним словом. Такая передача управления называется ближней (тип NEAR), а ее вариант с сокращённым диапазоном адресов переходов – короткой. Передача управления за пределы текущего сегмента кода называется межсегментной или дальней (тип FAR) – при этом необходимо модифицировать содержимое регистров IP и CS и адрес перехода представляется двумя словами (сегмент: смещение).

15) В системе команд процессора 8086 имеется 19 двухбайтных команд условных переходов. При выполнении этих команд анализируется некоторое условие, закодированное текущими состояниями флагов, и если оно выполняется, то осуществляется переход, а если нет, то выполняется следующая по порядку команда. Все условные переходы являются короткими. Некоторые команды для удобства программирования могут иметь несколько различных мнемонических обозначения.

16) Для управления циклами применяются три команды. В них предусматривается использование регистра CX в качестве счетчика цикла. В поле операнда команд управления циклами находится метка первой команды цикла (8-битовое смещение). Диапазон переходов этих команд составляет -128 - +127 байт от следующей команды. Команда LOOP производит декремент регистра CX и, если содержимое CX не равно нулю, происходит переход к началу цикла. В противном случае выполняется следующая по порядку команда. Мнемоники LOOPE/LOOPZ определяют одну и ту же машинную команду, которая производит декремент регистра CX, а затем передает управление в начало цикла, если содержимое CX не равно нулю и ZF=1. В противном случае выполняется следующая по порядку команда. Мнемоники LOOPNE/LOOPNZ также определяют одну и ту же машинную команду, которая производит декремент регистра CX, а затем передает управление в начало цикла, если содержимое CX не равно нулю и ZF=0. В противном случае выполняется следующая по порядку команда.