Сводный план 2016г., поз. 124

Коржымбаев Турсын Толебаевич

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕР В СРЕДЕ WINDOWS

Методические указания к выполнению лабораторных работ для студентов всех форм обучения специальности - 5В070400-«Вычислительная техника и программное обеспечение»

Редактор:

Подписано в печать _____ Формат 60x84 1/16
Тираж 150 экз. Бумага типографская №1
Обьём 2,4 уч. изд. л. Заказ ___ Цена ___

Копировально-множительное бюро
Некоммерческого акционерного общества
“Алматинский университет энергетики и связи”
050013, Алматы, ул. Байтурсынова, 126

Некоммерческое акционерное общество

АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ

Кафедра компьтерных технологии

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА В СРЕДЕ WINDOWS

Методические указания к выполнению лабораторных работ для студентов специальности -
5В070400 – “Вычислительная техника и программное обеспечение”

Алматы 2016

СОСТАВИТЕЛИ: Коржымбаев Т.Т. Программирование на языке Ассемблера в среде Windows : 5В070400 – Методическое указания к выполнению лабораторных работ для студентов специальности “Вычислительная техника и программное обеспечение” – Алматы: АИЭиС, 2015 – 52 б.

В этих методических указаниях приведены содержание и темы лабораторных работ. Рассмотрены способы и методы построения программ на языке Ассемблера. Методические указания предназначены для студентов специальности 5В070400 – “Вычислительная техника и программное обеспечение”.

Список литературы – 14 назв. .

Рецензент:

Печатается по плану некоммерческого акционерного общества «Алматинский университет энергетики и связи» 2016 года.

Введение

Язык Ассемблера является машинно-ориентированным языком, т.е. это язык процессора. Язык Ассемблера – это универсльный язык структурного программирования, являющийся удобным и гибким средством оптимизации кода программ, написания драйверов, трансляторов, защитных процедур, программирования некоторых внешних устройств и т.д..

Основной целью лабораторных работ – является обучение студентов базовым понятиям языка ассемблера, архитектуру компьютеров на основе процессоров Intel, основным аспектам современного программирования на ассемблере, включая системное и прикладное программирование.

Выполнение каждой лабораторной работы должно завершаться оформлением отчета. Отчёт о проделанной работе должен содержать :

- цель и задание работы;
- краткие итоги теорретической подготовки;
- результаты проделанной работы (тексты прграмм, результаты работы программ) ;
- список использованной работы.

Выполненная работа и оформленный отчёт защищается у преподавателя.

Лабораторная работа №1. Разработка программы с использованием разных группы команд процессора.

Цель работы: Обучение организации работы с отдельными битами, обучение свойствам цепочек команд и командам двойчно-десятичной арифметики.

Выполните следующие задания:

1)Напишите программу, считающую количество единиц в данном слове или байте:

;сегмент данных

A DB 01101011D или A DW 0010111110101001B

2)Измените программу так, чтобы она считала количество нулей.

3)Задав массив A состоящий из чисел со знаком, напишите программу считающую количество отрицательных чисел. Принимается любой вариант. Повторите это задание с использованием команды TEST.

4)Измените программу так, чтобы она считала количество положительных чисел.

5)Напишите программу преобразования двух неупакованных BCD-чисел в слове памяти упакованное BCD число в регистре AL.

6)Напишите программу переводящую символы из одной строки в другую:

; в сегменте данных

str_source DB ‘Переводимая строка’, ‘$’ ; строка-источник

str_dest DB 20dup(?) ; строка-приёмник

7)Напишите программу сравнивающую две строки по одному символу (одному элементу).

8)Напишите программу для поиска в данной строке одного символа. Поменяйте этот символ на заранее сохраненную в памяти или на записанную на клавиатуры.

9)Напишите программу для сравнения двух строк. Первый отличающийся элемент запишите в регистр AL.

10)Напишите программу вычисляющую по формуле:

$y=(a+b)^3/(c-d)^2$

Методические указания необходимые для выполнения лабораторной работы:

Команды сложения.

Команда сложения ADD. Формат написания команды:

ADD приёмник , источник

Команда сложения с переносом ADC. Формат записи команды:

ADC приёмник , источник

При выполнении команды сложения ADD результат записывается в операнд-приёмник. При выполнении команды сложения ADC результат записывается в операнд приёмник, но при сложении применяется флаг переноса. Также команды ADD и ADC могут использовать флаги: CF, PF, AF, ZF, SF, OF.

AAA – ASCII-коррекция результата сложения. Эта команда корректирует результат сложения для предоставления этого результата в коде ASCII. Она преобразует значение регистра AL в правильную неупакованную десятичную цифру.

Команда используется в следующем контексте:

ADD, AL, BL

AAA

DAA - десятичная коррекция результата сложения Команда DAA корректирует сложение для представления в десятичной форме. Она преобразует содержимое регистра AL в две правильные упакованные десятичные цифры. Используется в контексте:

ADD AL, BL

DAAКоманды AAA и DAA не требуют наличие операндов потому, что корректируемое значение находится в регистре AL.

Команда INC (увеличение на 1) увеличивает значение регистра или ячейки памяти на 1.

Формат написания команды: INC приёмник.

Команда удобна для приращения значений счетчиков в циклах, командах и значения индексного регистра или указателя при доступе к последовательно расположенным ячейкам памяти.

Эта команда меняет следующие флаги: PF, AF, ZF, SF, OF.

Команды вычитания.

Команда вычитания SUB. Формат команды: SUB приёмник, источник.

Команда SBB – вычитание с займом.

Команды SUB и SBB и форматы их написания похожи на команды и форматы команд сложения. Они тоже могут изменить шесть флагов.

AAS - ASCII – коррекция результата вычитания. Команда AAS корректирует результат вычитания для представления в кодах ASCII.

DAS - десятичная коррекция результата вычитания. Команда DAS корректирует результат вычитания для представления в десятичной форме.

Команды AAS и DAS похожи на команды AAA и DAA.

Команда используется в следующем контексте:

SUB AL, BL

AAS

или

SUB AL, BL

DAS

Команда DEC (уменьшение на 1) уменьшает значение ячейки памяти или регистра на 1.

NEG команда изменения знака операнда. Вычитает значение операнда из нулевого значения. Допустим, нужно вычесть из 100 значение регистра AL. Здесь, так как число 100 не идёт в качестве операнда-приёмника писменно нельзя писать так

SUB 100, AL

Если нужно организовать такое вычитание, нужно писать следующую команду:

NEG AL

ADD AL, 100

Эта команда изменяет следующие флаги: PF, AF, ZF, SF, OF и CF=1, если операнд ненулевое число, в других случаях они равны 0.

Команда CMP (сравнение) – в зависимости от результата вычитает приёмник из источника, обнуляет или устанавливает флаги, а значения приёмника и источника не меняются.

Эта команда меняет следующие флаги – PF, AF, ZF, SF, OF и CF.

Команды умножения.

Команда умножения чисел без знака MUL. Формат команды: MUL источник .

Команда умножения целых чисел со знаком IMUL. Формат команды: IMUL источник.

В качестве источника можно взять двоичное слово или слово, ячейку памяти размером в байт или регистр общего назначения.

В качестве операнда можно использовать регистры AL (для байта), AX (для слова) или EAX (для двоичного слова). Произведение имеет двойной размер и возвращается следующим образом: умножение байтов возвращает 16-битовое произведение в регистрах AH (старший байт) и AL (младший байт); умножение слов возвращает 32-битовое произведение в регистрах DX (старшее слово) и AX (младшее слово); умножение двойных слов возвращает 64-битовое произведение в регистрах EDX (старшее двойное слово) и EAX (младшее двойное слово).

Если после выполнения команды MUL большая половина результата произведения равна нулю, тогда CF=OF=0, в других случаях равны 1.

Если после выполнения команды IMUL большая часть результата произведения показывает лишь расширенный знак меньшей части, тогда CF=OF=0, в других случаях равны 1.

AAM – ASCII коррекция результата умножения. Команда преобразует результат предшествующего умножения байтов в два правильных неупакованных десятичных операнда.

Команды деления.

DIV - Команда деления без знака. Формат команды: DIV источник

IDIV - Команда деления чисел со знаком. Формат команды: IDIV источник

Делитель размером в байт, слово или двойное слово находится в регистре общего назначения или в ячейке памяти. Делимое должно иметь двойной размер, оно извлекает из регистров AH и AL (при делении на 8-битовое число), из регистров DX и AX (при делении на 16-битовое число) или из регистров EDX и EAX (при делении на 32-битовое число). Результат возвращается следующим образом: если источник представляет собой байт, то частное возвращается в регистр AL, а остаток в регистр AH. Если источник представляет собой слово, то частное возвращается в регистр AX, а остаток - в DX и если источник представляет собой двойное слово, то частное возвращается в регистр EAX, а остаток - в EDX.

AAD - ASCII коррекция результата деления. Исполняется непосредственно перед операцией деления. Команда преобразует неупакованное делимое в двоичное число и загружает его в регистр AL.

Логические команды и команды сдвига.

Логические команды и команды сдвига позволяют манипулировать битами. Команды сдвига позволяют выполнить быстрое умножение и деление операндов на степени двойки, а также эффективное преобразование данных. Логические команды выполняют операции над операндами размерностью байт, слово или двойное слово

AND – команда логического умножения И (конъюнкция) над битами операндов приемник и источник.

Формат команды: AND приемник, источник

OR - команда логического сложения ИЛИ (дизъюнкция) над битами операндов приемник и источник.

Формат команды: OR приемник, источник

XOR - команда логического исключающего сложения ИЛИ над битами операндов приемник и источник. Формат команды: XOR приемник, источник

Результат вычисляется в соответствии с таблицей истинности логических операций:

Приёмник	Источник	AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

TEST – (“test – проверить”) команда логического сравнения посредством логического умножения И (конъюнкция) над битами операндов приемник и источник, но состояние операндов остается прежним, изменяются только флаги zf, sf, и pf, что дает возможность анализировать состояние отдельных битов операнда без изменения их состояния.

Формат команды: **TEST приемник, источник**

NOT - команда логического отрицания каждого бита операнда источник.

Формат команды: NOT источник. Команда инвертирует все биты операнда источник: из 1 в 0, из 0 в 1. Выполнение команды не влияет на флаги**.**

Примеры:

1) mov AL, 1111 0000b

2) mov AL, 1111 0000b
3) mov AL, 1111 0000b and AL, 1010 1010b or AL, 1010 1010b xor AL, 1010 1010b
   результат 1010 0000b; результат 1111 1010b; результат 0101 1010b
4) mov AL, 11110000b
5) проверить число на четность: not AL; A:=00001111b test AL, 0000 0001b
   jz m; не чётное число m; чётное число

Как показано в примере команда AND устанавливает бит приёмника в 0, соответственно бит источника равен нулю, команда OR – устанавливает бит приёмника в 1, соответственно биты источника равны 1, команда XOR – инвертирует биты приёмника, соответственно биты источника также инвертируются.

Команды сдвига.

По принципу действия команды сдвига можно разделить на два типа:

- команды линейного (логического SHL, SHR или арифметического SAL, SAR) сдвига;

- команды циклического (простого ROL, ROR и через флаг переноса RCL, RCR) сдвига.

Все команды сдвига перемещают биты в поле операнда влево или вправо в зависимости от кода операции. Все команды сдвига имеют одинаковый формат: первым операндом является операнд-источник, вторым операндом является количество сдвигов. Количество сдвигов либо указывается непосредственно в команде, либо перед выполнением команды сдвига заносится в регистр CL.

Например,

MOV CL, 4 .386

SHL AX, 1 SHL AX, CL SHL AX, 4 (для больших процессоров)

SHL – команда логического сдвига влево сдвигает содержимое операнда влево на количество битов, определяемое значением счетчик_сдвигов. Справа (в позицию младшего бита) записываются нули.

SHL операнд, счетчик_сдвигов

SHR - команда логического сдвига вправо сдвигает содержимое операнда вправо на количество битов, определяемое значением счетчик_сдвигов. Слева (в позицию старшего бита) записываются нули.

SHR операнд, счетчик_сдвигов

SAL – команда арифметического сдвига влево сдвигает содержимое операнда влево на количество битов, определяемое значением счетчик_сдвигов. Справа (в позицию младшего бита) записываются нули.

SAL операнд, счетчик_сдвигов

Команда SAL не сохраняет знака. Команда SAL полностью аналогична команде SHL.

SAR – команда арифметического сдвига вправо сдвигает содержимое операнда вправо на количество битов, определяемое значением счетчик_сдвигов. Слева (в позицию старшего бита) записываются нули, если сдвигается положительное число и единицы, если сдвигается отрицательное число.

SAR операнд, счетчик_сдвигов

Команда **SAR** сохраняет знак, восстанавливая его после сдвига каждого очередного бита.

ROL – команда циклического сдвига влево сдвигает содержимое операнда влево на количество битов, определяемое значением счетчик_сдвигов. Сдвигаемые влево биты записываются в тот же операнд справа.

ROL операнд, счетчик_сдвигов

ROR – команда циклического сдвига вправо сдвигает содержимое операнда вправо на количество битов, определяемое значением счетчик_сдвигов. Сдвигаемые вправо биты записываются в тот же операнд слева.

ROR операнд, счетчик_сдвигов

RCL – команда циклического сдвига влево через флаг переноса сдвигает содержимое операнда влево на количество битов, определяемое значением счетчик_сдвигов. Сдвигаемые биты поочередно становятся значением флага переноса CF. Одновременно старое значение флага переноса CF вдвигается в операнд справа и становится значением младшего бита операнда.

RCL операнд, счетчик_сдвигов

RCR – команда циклического сдвига вправо через флаг переноса сдвигает содержимое операнда вправо на количество битов, определяемое значением счетчик_сдвигов. Старое значение флага переноса CF вдвигается в операнд слева и становится значением старшего бита операнда

RCR операнд, счетчик_сдвигов

Команды обработки строк

Система команд микропроцессора имеет очень интересную группу команд, позволяющих производить действия над блоками элементов от 1 Байта до 64 Кбайт (для 32-разрядных микропроцессоров – до 4 Гбайт). Это - цепочечные команды или команды обработки строк символов. Эти блоки логически могут представлять собой последовательности элементов с любыми значениями, хранящимися в памяти в виде двоичных кодов. Микропроцессор всегда предполагает, что строка-приемник находится в дополнительном сегменте (адресуемом посредством сегментного регистра ES), а строка-источник – в сегменте данных (адресуемом посредством сегментного регистра DS). Микропроцессор адресует строку - приемник через регистр DI, а строку-источник - через SI. Допускается переопределять сегмент для строки_источника, для строки-приемника этого делать нельзя.

Особенность работы цепочечных команд состоит в том, что они автоматически выполняют приращение или уменьшение содержимого регистров DI и SI в зависимости от используемой цепочечной команды. Что именно происходит с этими регистрами, определяется состоянием флага DF, которым управляют команды CLD и STD.

Всего в системе команд микропроцессора имеется пять операций-примитивов обработки цепочек. Каждая из них реализуется в микропроцессоре двумя командами (для для 32-разрядных микропроцессоров - тремя командами), в свою очередь, каждая из этих команд работает с соответствующим размером элемента – байтом, словом или двойным словом.

Пересылка строк:

MOVS адрес приёмника => MOVSB
MOVSW
адрес источника MOCSD

Сравнение строк:

CMPSB
CMPS адрес приёмника, адрес источника =>CMPSW
CMPSD

Поиск в строке:

SCASB
SCAS адрес приёмника => SCASW

SCASD

Загрузка элемента строки:

LODSB

LODS адрес источника => LODSW

LODSD

Сохранение элемента в строке:

STOSB

STOS адрес приёмника => STOSW

STOSD

Контрольные вопросы:

1) Где размещаются множитель и результат умножения при выполнении команды умножения?
2) Где размещаются (делимое), делитель и результат деления при выполнении команды деления?
3) В каком регистре сохраняется длина строки в цепочечных командах?

Лабораторная работа №2. Выполнение простейших задач

Цель работы: ассемблирование програмы, обучение процессам компоновки и выполнения, а также обучение тестированию и отладке простейших типовых задач на ЭВМ. Как правило, в самой постановке такой задачи уже определен алгоритм ее решения. Этот алгоритм нужно написать с использованием конструкций языка Ассемблера. Для проверки правильности результата должны быть представлены необходимые тестовые данные.

Выполните следующие задания:

1)Выведите сообщение на экран с начала строки;

2)Выведите сообщение на экран в середине строки;

3)Выведите сообщение на экран с начала строки в рамке составленной из любых псевдографических символов.

4)Повторите предыдущее задание, но в середине строки.

5)Выведите на экран символ с помощью функции 2h. Для этого запишите в сегмент кода следующее:

```
mov   ah, 2h          ;функция вывода символа на экран
mov   dl, 'A'         ;ASCII
int   21h       ;прерывание DOS
```

6)Перед выводом сообщения почистите экран функцией 6 int 10h;

```
mov   ah, 6h          ;функция очистки экрана
mov   al, 0           ;0 – весь экран
mov   ch, 0           ; номер строки левого верхнего угла
mov   cl, 0           ; номер столбца левого верхнего угла
mov   dh, 24          ; номер строки правого нижнего угла
mov   dl, 79          ; номер столбца правого нижнего угла
mov   bh, 0           ; байт атрибут (на бирюзовом фоне черные символы)
int   10h             ;прерывание BIOS
```

Поместите эти 8 команд после 9 строки

7)Перед выводом сообщения поместите курсор в заданную позицию с помощью функции 2 int 10h:

```
mov   ah, 2h          ;функция для перемещения курсора
mov   bh, 0           ;текущая видеостраница
mov   dh, 5           ;номер строки – 5
mov   dl, 10          ;номер столбца – 10
int   10h             ;прерывание BIOS
```

Применяйте эти команды перед выводом сообщения или символа.

8)Перед выводом сообщения с помощью функции 6 команды int 10h нарисуйте разноцветное окошко и с помощью функции 2 команды int 10h разместите курсор.

Необходимые методические указания для выполнения лабораторной работы.

Введите в компьютер с помощью любого редактора первоначальный текст программы на ассемблере. При использовании редактора NC: F4 – редактирование файла обозначенного курсором; SHIFT+F4 – создание нового файла на диске, нужно вводить имя файла; ALT+F4 – внешний редактор.

Введите показанную программу:

```
Stacksg         segment
                dw      32dup(?)
Stacksg         ends
Datasgt         segment
string          db      ‘Имя и фамилия’, 13,10
                db      ‘Возраст: ....’, ‘$’
datasg          ends
codesg          segment
                assume          cs:codesg, ds:datasg,ss:stacksg
start:
                mov     ax, datasg
                mov     ds, ax
                mov     dx, offset string
                mov     ah, 9h
                int     21h
                mov     al, 0
                mov     ah, 4ch
                int     21h
  codesg        Ends
                end     Start
```

После отладки убедитесь что программа есть в текущем каталоге – Name.asm . Чтобы получить модуль .OBJ в командной строке нужно набрать транслятор и имя вашего файла – TASM.exe NAME.asm .

Способ быстрого выполнения:

- на панели выделите файл TASM.exe и нажмите CTRL+ENTER;
- на панели выделите файл NAME.asm и нажмите CTRL+ENTER;

Проверьте информация в командной строке. Для выполнения трансляции жмите ENTER.

Если выводятся сообщения об ошибках, исправьте исходный код программы с помощью редактора и выполните трансляцию снова.

Для получения исполняемой программы в командной строке нужно набрать имя компоновщика TLINK.EXE и имя вашего обьектного файла NAME.obj:

TLINK.exe NAME.obj

Жмите ENTER.

В текущем каталоге появляется файл NAME.exe и NAME.map содержащая таблицу со сведениями об именах и размерах сегментов.

Выберите курсором исполняемый файл и нажмите ENTER. Такой исполнение программы возможно при явных результатах и при полной уверенности в её безошибочной работе.

При составлении программы на языке Ассемблера можно выделить следующие этапы:

1)Создание блок-схемы;

2)Составление исходной программы NAME.asm. NAME – любое название файла.

3)Создание обьектной программы NAME.obj.

4)Создание исполняемой программы NAME.exe.

5)Выполнение .exe программы.

6)Проверка результата выполнения программы.

Если результат выполнения программы не соответствует требованиям задачи, нужно искать ошибки и исправлять их.

Исходный текст программы составляется в любом текстовом редакторе. Редактор текста представляет собой программу, которая обеспечивает ввод и корректировку исходных текстов программ в кодах ASCII.

Транслятор (компилятор) NAME.obj (текст программы на машинном языке, но без связей между подпрограммами) формирует дисковый файл с обьектным модулем.

Компоновщик (загрузчик) осуществляет компоновку результирующих модулей с предполагаемыми библиотечными модулями и завершает определение адресных ссылок, т.е. создает перемещаемый исполняемый модуль NAME.EXE. Представленный здесь Турбо Ассемблер, включающий встроенные компилятор, компоновщик и отладчик, разработан фирмой Borland International для семейства

микрокомпьютеров IBM PC. Он совместим с ассемблером MASM фирмы Microsoft Corporation, но кроме того содержит много полезных расширений и улучшений и обладает более высоким быстродействием.

Компилятор Турбо Ассемблера – это выполняемая программа, размещенная в файле TASM. EXE. Компилятор вызывается командой TASM.exe NAME.asm, которая создаёт перемещаемый обьектный файл NAME.obj. В этой команде NAME.asm – это имя (транслируемого) файла.

В общих случаях вызов компилятора происходит следующим образом:

TASM fileset;filset;///;fileset

здесь fileset такой:

options, sources, object, list, xref

где options – имена опции, sources – имя исходного файла или цепочка строка имён файлов разделённых либо символом + (плюс) либо пробелом, object – имя полученного файла .obj, list - имя файла с листингом компиляции, xref – имя файла с таблицами перекрёстных ссылок. В таблице перекрёстных ссылок даны символьные имена модуля, мест, в которых они определены и номера строк, в которых появились ссылки на них.

Например, команда

TASM /1 prim1+prim2,prim,/zi test

создаёт модуль prim.obj из исходных модулей prim1.asm и prim2.asm, также листинга компиляции, который будет помещен в файле prim.lst (опция /l), после чего из исходного модуля test.asm с помощью опции /zi создаётся модуль test.obj.

Чтобы получить исполняемую программу нужно осуществить сборку получившихся модулей и библиотечных модулей. Эту задачу выполняет компоновщик Turbo Link фирмы Borland Int..

Компоновщик является выполняемой программой, хранящейся в файле TLINK.EXE.

В общих случаях вызов сборщика TLINK выполняется следующим образом:

TLINK options, objects, exec, map, libraries

здесь options – набор опции, objects – набор имён файлов с получившимися модулями, exec – имя файла, в который записывается исполняемая программа, map – имя файла, где находится карта

компоновки, libraries - набор имён файлов, в которых хранятся библиотеки получившихся модулей. Опции разделяются пробелами, имена файлов разделяются либо пробелами либо символом + (плюс).

Контрольные вопросы:

1)Структура программы на языке Ассемблер.
2)Этапы выполнения программы.
3)Функции вывода 9h и 2h.
4)Как создаётся исходный файл?
5)Расширение исходного файла.
6)Имя транслятора.
7)Что делает транслятор?
8)Какие файлы получаются на выходе трансляции?
9)Пример командной строки запуска транслятора.
10)Имя компоновщика.
11)Какие файлы являются входными во время компоновки?
12)Какие файлы являются выходными во время компоновки?
13)Какое расширение у исполняемого файла?
14) Пример командной строки запуска компоновщика.
15)Как можно просмотреть файл листинга?

Лабораторная работа №3. Передача управления программе

Цель работы: Обучиться методам программирования нелинейных алгоритмов.

Выполните следующие задания:

1)Перед выводом массива B на экран выведите следующую текстовую строку: «Это второй массив B». Т.е. на экране должна быть следующая строка:

Это второй массив B 1 2 3 4 5

Выведите элементы массива через пробел (или запятую).

2)Измените количество элементов в массиве, и введите нужные изменения в программу.

3)В теле программы измените программу через организацию одного цикла.

4)Переделайте данную программу в программу пересылки массива слов.

5)Напишите программу перевода массива байтов в массив слов.

```
;сегмент данных
A       db    1,2,3,4,5,6
B       dw    3dup(?)
;сегмент кода
(ваша программа)
```

6)Напишите программу сложения двух байтовых массивов A и B. Результат сохраните в C массив.

```
; сегмент данных
A       db    1,2,3,4
B       db    5,6,3,1
C       db    4dup(?)
```

7)Даны массивы байтов A и B. В третий массив нужно записать максимальные элементы (сравнивая поэлемент). Принимается любой ваш вариант.

```
;сегмент данных
A       db    1,2,3,4
B       db    5,6,3,1
C       db    4dup(?)
```

После выполнения вашей программы на экране должен появится следующий результат:

Массив C 7 8 9 6

Желаем удачи!

8)Переделайте предыдущую программу так, чтобы она искала и записывала в массив C минимальные элементы. Т.е. результат должен быть следующим:

Массив C 4 5 2 3

Вперёд!

9)Теперь напишите программу выполняющую поиск максимального элемента среди элементов данного массива байтов. Например,

```
;сегмент данных
A       DB   9,8,7,6,5,4
```

В результате получится следующее: максимальный элемент массива A – элемент 9.

А этот результат - последствие выполнения вашего EXE файла.

10) Переделайте предыдущую программу так, чтобы выполнялся поиск минимального элемента среди элементов данного массива байтов. Конечно, результат будет другим:

Минимальный элемент массива A – элемент 4.

Если результат на экране не меняется, то вы допустили ошибку.

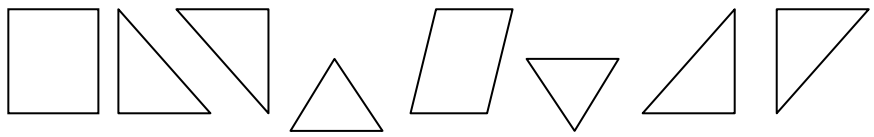
11)Допустим, дан массив A.

```
A       DB   1,1,-2,-4,6
        DB   2,-5,2,-2,5
        DB   3,-3,-3,4,4
        DB   5,-5,6,-7,7
```

В данной области памяти нужно заменить все отрицательные числа на 0.

```
1 1 0 0 6
2 0 2 0 5
3 0 0 4 4
5 0 6 0 7
```

12)Нарисуйте следующие фигуры звёздочками (*) в текстовом режиме с помощью функции INT 21h 2:



13)Напишите процедуру вывода на экран в двоичном коде.

14) Напишите процедуру вывода на экран в шестнадцатеричном коде.

Необходимые методические указания для выполнения лабораторной работы.

Рано или поздно любая программа прекращает своё выполнение, и появляются точки, где необходимо решить куда и как передать управление. Иногда такие ситуации могут быть связаны с каким-нибудь условием, а иногда бывает просто необходимо “уйти” в другое место. В языке Ассемблер место, куда необходимо передать управление, определяется меткой. Метка – это символьное название, обозначает определённую ячейку памяти, в командах передачи управления используется как операнд.

Команды передачи управления делятся на группы:

- команды организации цикла
- команды безусловной передачи управления;
- команды условной передачи управления.

Команда организации цикла

Одной из команд организации цикла является команда LOOP. Количество повторений цикла задается значением в регистре CX перед входом в последовательность команд, составляющих тело цикла. Команда LOOP выполняет декремент содержимого регистра CX, и если оно не равно 0, осуществляет переход на указанную метку вперёд или назад в том же программном сегменте в диапазоне -128...+127байт.

Команды безусловной передачи управления:

1)Команда безусловной смены – JMP метка.

2)Команды вызова процедуры и возвращения из процедуры – CALL имя процедуры и RET;

3)Вызов программных прерывании и – INT номер_прерывания, IRET – возвращение из программных прерываний, INTO – прерывание в случае переполнения.

Команды условной передачи управления.

CMP операнд1, операнд2

Алгоритм работы:

- вычитание (операнд1- операнд2);
- установка флагов в зависимости от результата, операнд1 и операнд2 не должны меняться (т.е. нет необходимости сохранения результата).

JE Операнд1=операнд2 JNE операнд1<>операнд2

Без знака Со знакомКритерии условной смены

JB/JNAE JL/JNGE операнд1<операнд2
JBE/JNA JLE/JNG операнд1<=операнд2
JA/JNBE JG/JNLE операнд1>операнд2
JAE/JNB JGE/JNL операнд1>=операнд2

2)Команды перехода по состоянию определённого флага

JC (JNC) Метка_перехода
JP (JNP) Метка_перехода
JZ (JNZ) Метка_перехода
JS (JNS) Метка_перехода
JO (JNO) Метка_перехода

3)Команда перехода зависящая от содержимого регистра CX

JCXZ Метка_перехода

Порядок выполнения работы

Дана программа пересылки массива байтов A на место массива байтов B.

```
sseg   segment
       Db      128dup(?)
sseg   ends
dseg   segment
A      Db      1,2,3,4,5
B      Db      5 dup(?)
dseg   ends
cseg   segment
       assume      ss:sseg,cs:cseg,ds:dseg
 start:
       mov  ax, dseg
       mov  ds, ax
       mov  si, 0
       mov  cx,5
 MI:
       mov  al, A[si]
       mov  B[si], al
       inc  si
       loop M1
```

```
        mov  cx, 5
        mov  si, 0
M2:
        mov  ah,2h
        mov  dl, B[si]
        mov  21h
        inc  si
        loop M2
        mov  ah, 4ch
        int  21h
cseg    ends
        end  start
```

Контрольные вопросы:

1)Какие команды условного перехода используются при операциях над числами со знаком?

2)Какие безусловные команды смены используются для обхода группы команд?

3)Какие команды используются для организации цикла?

4)Какие команды выполняют вызов программы, обработку прерывания, и возвращение из неё? (?)

5)Какие команды выполняют вызов процедуры (внутренней программы) и возвращение из неё?

Лабораторная работа №4. Описание простых типов данных. Отладчик Turbo debugger (TD)

Цель работы: Обучение правилам описания простых типов данных и основным моментам работы отладчика TD.

Выполните следующие задания:

1)Наберите исходный текст программы в соответствии с вариантом:

```
Datasg      segment
Mess        db          ‘Директивы данных $’
Pa          db
Pb          dw
Pc          dd
```

```
Mas         db          данные в (вариантах) задания
Pole        db
Adr         dw
Adr_full        dd
Datasg          ends

codesg          segment
                assume          cs:codesg, ds:datasg
start:
                mov         ax, datasg
                mov         ds, ax
                mov         dx, offset mess
                mov         ah, 9h
                int         21h
                mov         ax, 4c00h
                int         21h
 codesg         ends
                end         Start
```

После получения загружаемого модуля, его нужно запустить отладчик Turbo (TD). В окошке DUMP посмотрите сегмент данных, найдите все переменные, показанные в вашем варианте, и обьясните места расположения и обьём этих переменных. Вы отвечаете не только за описание переменной в сегменте данных, но также и за каждый байт памяти.

2)В сегменте кода наберите вот эти команды:

```
mov   AL,  Pa               ;al=?
mov   Bx,  Pb               ;bx=?
mov   Bl,  byte ptr Pb      ;bl=?
mov   Dx,  word ptr Pc      ;dx=?
mov   Cx,  word ptr Pc+2    ;cx=?
mov   Dl,  byte ptr Pc      ;dl=?
mov   Dh,  byte ptr Pc+1    ;dh=?
```

Посмотрите результат выполнения этих команд в отладчике.

Методические указания необходимые для выполнения лабораторной работы.

TASM предоставляет широкий набор инструментов для описания и обработки данных . Его можно сравнить с похожими инструментами некоторых высокоуровневых языков. Правила описания простых типов данных являются базовыми для описания более сложных типов. Для описания простых типов данных в программе используются специальные директивы резервирования и инициализации данных, которые по сути являются указаниями транслятору на выделение определенного объема памяти. TASM применяет следующие директивы данных:

DB(Define Byte) - резервирование места в памяти для данных обьёмом 1 байт;

DW(Define Word) - резервирование места в памяти для данных обьёмом 2 байт;

DD(Define Double Word) - резервирование места в памяти для данных обьёмом 4 байт;

DF(Define Far Word) - резервирование места в памяти для данных обьёмом 6 байт;

DP(Define Pointer) - резервирование места в памяти для данных обьёмом 6 байт;

DQ(Define Quarter Word) - резервирование места в памяти для данных обьёмом 8 байт;

DT(Define Ten Bytes) - резервирование места в памяти для данных обьёмом 10 байт;

Директивы данных имеют следующий формат :

[Название] мнемоника директивы [(выражение)] [;комментарии]

Для того, чтобы определить переменную без присваивания ей значения, нужно в поле выражения поставить знак вопроса (?). Например,

A db ? – не инициализированная переменная A (содержимое выделенного участка физической памяти обьёмом 1 байт не меняется).

A db 10011011b - инициализированная переменная (транслятор выделяет в памяти место на 1 байт для переменной A и записывает в него нужное значение).

Правило расположения данных в памяти напрямую связан с логикой работы микропроцессора с данными. Микропроцессоры Intel ставит следующий принцип расположения данных в памяти: младший байт в младшем адресе.

Отладчик Turbo Debugger (TD) представляет собой оконную среду отладки программ на уровне исходного текста на многих языках программирования и, в том числе на ассемблере. Он даёт возможность найти место логической ошибки и причины её возникновения.

В основной части окна отладчика есть одно или несколько окошек. В каждый момент только один из них может быть активен. Чтобы активировать любое окошко нужно щёлкнуть мышкой на любой точке окошка. С помощью системы меню можно управлять работой отладчика. Есть два вида такого меню:

- глобальное меню – расположено в верхней части окна отладчика. К нему можно обращаться в любой момент клавишей F10;

- локальное меню – в каждом окошке отладчика можно вызвать его собственное меню, которое учитывает осбенности этого окна (ALT+F10).

Среди четырёх режимов вы будете часто использовать режим пошагового выполнения программы. В этом режиме вы можете выполнять по одной команде и видеть результат работы каджой команды. Чтобы активировать этот режим нужно нажать F7 (процедуры выполняются по шагам) или F8 (процедура обрабатывается как одна команда). В этом режиме полезно использовать окошко CPU, его можно вызвать через меню VIEW|CPU.

Это окошко показывает состояние микропроцессора и состоит из 5 внутренних окошек:

- Окошко, показывающее исходную программу в дизассемблированном виде. (это – та же программа, что и в окне MODULE , но в виде машинного кода; пошаговую отладку можно производить прямо в этом окне).

- REGISTERS – окошко регистров микропроцессора, показывающее текущее состояние регистров;

- окошко стека STACK, показывающее содержимое памяти отданное стеку.

- окошко с дампом памяти DUMP. Оно показывает содержимое памяти в области данных.

В окошке отладчика под названием MODULE вы увидите исходный текст программы с курсором выполнения (в виде треугольника).

Исходный модуль нужно транслировать опцией /zi:

Tasm /zi имя_исходного_модуля

(Сборку) модуля нужно выполнять опцией /v:

Tlink /v имя_обьектного_модуля

Запуск отладчика производить из командной строки с указанием исполняемого модуля программы: Td имя_выполняемого_модуля.

Варианты заданий:

1)	Pa	db	73H	9)	Pa	db	5BH
	Pb	dw	0AE21H		Pb	dw	0BA21H
	Pc	dd	38EC76A4H		Pc	dd	0FA4A32BCH
	Mas	db	10 dup(1),2,3		Mas	db	4,5,6,5 dup(0)
	Pole	db	5 dup(?)		Pole	db	6 dup (?)
	Adr	dw	Pc		Adr	dw	Pc
	Adr_full	dd	Pc		Adr_full	dd	Pc
2)	Pa	db	67H	10)	Pa	db	4AH
	Pb	dw	4AEFH		Pb	dw	0DEFCH
	Pc	dd	12DC4567H		Pc	dd	81ADFF06H
	Mas	db	5,6,7,8		Mas	db	5 dup(1),2,3,3 dup(4)
	Pole	db	6 dup(0)		Pole	db	6 dup(« «)
	Adr	dw	Pc		Adr	dw	Pc
	Adr_full	dd	Pc		Adr_full	dd	Pc
3)	Pa	db	4DH	11)	Pa	db	7FH
	Pb	dw	0ED56H		Pb	dw	0ACDEH
	Pc	dd	32AF8DD7H		Pc	dd	10B0A488H
	Mas	db	4,3,5,4 dup(0)		Mas	db	3 dup(0),1,2,3, 4 dup(0)
	Pole	db	6dup(?)		Pole	db	5dup(32)
	Adr	dw	Pc		Adr	dw	Pc
	Adr-full	dd	Pc		Adr_full	dd	Pc
4)	Pa	db	5DH	12)	Pa	db	0BCH
	Pb	dw	0A1A3H		Pb	dw	903FH
	Pc	dd	3 dup(4),5,6		Pc	dd	6CAA3E41H

	Mas	db	4,3,5,4 dup(0)		Mas	db	1,2,3,4 dup(4)
	Pole	db	5 dup(?)		Pole	db	5 dup(?)
	Adr	dw	Pc		Adr	dw	Pc
	Adr_full	dd	Pc		Adr_full	dd	Pc
5)	Pa	db	62h	13)	Pa	db	0FBH
	Pb	dw	7ED1H		Pb	dw	54ADH
	Pc	dd	0EE45DA31H		Pc	dd	0E04365FAH
	Mas	db	1,2,6 dup(3),0		Mas	db	3 dup (0), 4 dup(1),2,3
	Pole	db	5dup(0)		Pole	db	5 dup (?)
	Adr	dw	Pc		Adr	dw	Pc
	Adr_full	dd	Pc		Adr_full	dd	Pc
6)	Pa	db	0FFH	14)	Pa	db	11H
	Pb	dw	4ADEH		Pb	dw	4D2DH
	Pc	dd	0C23891F5H		Pc	dd	98ADF156H
	Mas	db	4 dup (0), 1, 2, 3		Mas	db	5 dup(0), 1, 2, 3
	Pole	db	3 dup(‘’)		Pole	db	3 dup(?)
	Adr	dw	Pc		Adr	dw	Pc
	Adr_full	dd	Pc		Adr_full	dd	Pc
7)	Pa	db	0AEH	15)	Pa	db	10H
	Pb	dw	63BCH		Pb	dw	1A2DH
	Pc	dd	63BCDEF3H		Pc	dd	55AEF2C8H
	Mas	db	9,8,3 dup(0)		Mas	db	1, 2, 3, 4, 5, 6
	Pole	db	5 dup(«»)		Pole	db	5 dup (0)
	Adr	dw	Pc		Adr	dw	Pc
	Adr_full	dd	Pc		Adr_full	dd	Pc
8)	Pa	db	67	16)	Pa	db	89H
	Pb	dw	4AEFH		Pb	dw	91ADH
	Pc	dd	12DC4567H		Pc	dd	0AFD43e55H
	Mas	db	5, 6, 7, 8		Mas	db	5, 6, 3 dup (9)
	Pole	db	6 dup(0)		Pole	db	4 dup (?)
	Adr	dw	Pc		Adr	dw	Pc
	Adr_full	dd	Pc		Adr_full	dd	Pc

Контрольные вопросы:

1)Как размещаются в памяти байты?

2)Как размещаются в памяти слова и двоичные слова? (???)

3)С какого байта адресуются слова и двоичные слова?

4)Как размещается в памяти полный адрес переменной?

5)Если размеры переменной и регистра разные, какие операторы применяются в команде ассемблера?

Лабораторная работа №5. Использование системных функций в прикладной программе

Цель работы: ввод с клавиатуры, вывод на экран, работа с файлами с помощью инструментов BIOS.

Выполните следующие задания:

Нужно написать программу:

1. Ввода строки с клавиатуры с помощью ф.0Ah int 21h (ф.3Fh int 21h);
2. Ввода пароля (с помощью ф.0 int 16h, ф.7, ф.8 int 21h) и проверки его;
3. Проверки нажатия клавиши Enter (F1, Home);
4. Организации паузы и продолжения программы по нажатию клавиши (любой, конкретной);
5. Ввода строки и замены байта 0Dh на 0;
6. Ввода строки с клавиатуры с помощью ф.1 int 21h, ф.0 int 16h.
7. Вывода символа с помощью ф.2 int 21h (ф.9 int 10h);
8. Вывода группы одинаковых символов с помощью ф.21 int 21h (ф.9 int 10h);
9. Вывода массива на экран;
10. Вывода горизонтальной линии с помощью ф.10 int 10h (ф.14 int 10h, ф.9 int 10h);
10. Вывода текстовой строки с помощью ф.9 int 10h (ф.40 int 21h);
12. Вывода рамки с помощью ф.9 int 21h (ф.40h int 21h, ф.2 int 21h, ф.10 int 10h).
13. Записать в файл текстовую строку.
14. Определить длину файла.
15. Прочитать файл и вывести его содержимое на экран.
16. Поменять в файле 10-й байт.
17. Прочитать в файле 5-й байт и вывести его на экран.
18. Скопировать один файл во второй.
19. Скопировать файл начиная с 20-го байта во второй.

20. Прочитать файл и записать в последний байт контрольную сумму.
21. Сравнить два файла на идентичность.
22. Объединить два файла.
23. Определить номер заданного байта.
24. Переименовать файл и установить в нем время и дату создания.
25. Переименовать файл и поменять в нем атрибут.

Методические указания необходимые для выполнения лабораторной работы.

Обзор системных функций ввода с клавиатуры.

Функции ввода с клавиатуры выполняются с помощью прерываний INT 16h (функции 0,1) и INT 21h (1,7,8, 0Ah, 3Fh).

Функция 00h INT 16h ждёт нажатия клавиши.

После нажатия клавиши в регистр AL записывается либо ASCII код клавиши (если нажата клавиша с ASCII кодом) либо 0 (если нажата функциональная клавиша). А в AH регистр записывается скан-код клавиши.

По значению регистра AL можно определить была ли нажата символьная клавиша или функциональная.

```
KEY: MOV   AH, 0     ;ожидание нажатия клавиши
     INT   16h       ;вызов BIOS
     CMP AL, 0       ;проверка нажатия
     JE    M1        ;функциональной клавиши
     JMP  KEY        ;не функциональная клавиша
M1:  ...             ;функциональная клавиша
```

Функция 1h INT 16h осуществляет чтение информации о состоянии буфера клавиатуры; если буфер пуст, возвращает во флаге нуля 1; если буфер не пуст, возвращает во флаге нуля 0; в AX - очередной символ, остальные - в буфере.

Функция 01h INT 21h выполняет ввод символа с клавиатуры. Выводит это символ на экран и выполняет проверку клавиш Ctrl-Break. После ввода символа в AL сохраняется ASCII код символа.

```
MOV AH, 1h
INT   21h
```

Функция 07h INT 21h выполняет ввод символа с клавиатуры без вывода его на экран. После выполнения функции в регистре AL сохраняется ASCII код символа.

```
MOV AH, 7h
INT  21h
```

Функция 08h INT 21h похожа на функцию 07h, но проверяет клавиши Ctrl-Break.

```
MOV AH, 8h
INT  21h
```

Для того, чтобы прочесть функциональные клавиши с помощью функции 01, 07, 08 нужно выполнить две операции чтения кода клавиши. Первая операция чтения в регистр AL 0, вторая операция чтения вводит в регистр AL скан-код (расширенный код).

```
KEY: MOV      AH, 8h          ;ожидание нажатия клавиши
     INT  21h
     CMP AL, 0               ;расширенный ли код?
     INZ  ERROR    ;если нет, то сообщить об ошибке
     MOV      AH, 8          ;чтение скан-кода
     INT  21h
     CMP AL, 33h    ;нажата ли клавиша F1
     JE    F1       ;да
     JMP KEY        ;нет
     ERROR:  .....
```

Функция 0Ah INT 21h выполняет ввод строки с клавиатуры в буфер. Адрес буфера загружается в регистр DX. В сегменте данных нужно описать буфер и указать его размер. Первый байт буфера и должен содержать размер. После выполнения функции второй байт будет содержать фактическую длину вводимой строки, которая заканчивается кодом возврата каретки.

```
; в сегменте данных
BUF   DB   22,21 DUP(?)
;в сегменте кода
MOV AH,  0Ah
LEA   DX,  BUF
INT    21h
```

Ведённая строка в памяти начинается с адреса BUF+2.

Функция 3Fh INT 21h выполняет ввод данных с устройства. В качестве дескриптора устройства нужно задавать 0 (стандартный ввод - клавиатура). В регистре CX показывается количество ввводимых байтов, в регистр DX загружается адрес буфера. После выполнения функции в AX сохраняется точное число введённых байтов.

```
; в сегменте данных
BUF   DB   20 DUP (?)
; в сегменте кода
MOV AH, 9FH
MOV BX, 0                ;дескриптор клавиатуры
MOV CX, 20
LEA   DX, BUF
INT    21h
MOV CX, AX               ;сохранение в CX точного числа
введённых байтов
```

Для ввода пароля можно использовать любую из приведённых выше функции. Для ввода скрытого пароля используются функции вводящие символы без их вывода на экран.

Обзор системных функций вывода на экран.

Вся информация, выводимая на экран должна быть показана в ASCII кодировке. Для вывода символов на дисплей - BIOS обслуживает 3 функции (9, 10, 14) .

Функция 9 Int 10h – выводит символ с атрибутами в текущей позиции курсора. Во время вызова функции: BH – номер видеостраницы, (текстовый режим: 0- текущая страница), BL – атрибут символа, CX – количество повторении выведенных символов, AL – выводимый символ.

```
MOV AH, 9H
MOV BH, 0
MOV BL, 07h
MOV CX, 1
MOV AL, ‘A’
INT    10h
```

Функция INT 10h 10 - выводит символ без изменения атрибутов в текущую позицию курсора. Во время вызова функции: BH - номер видеостраницы, 0 – текущая страница, CX – количество повторении выведенных символов, AL – выводимый символ.

```
MOV AH, 10
MOV BH, 0
MOV CX, 1
MOV AL, ‘A’
INT  10h
```

Во время применения функций 09h и 10h, нужно помнить следующее. Эти функции не сдвигают курсор во время вывода символа (или группы символов).

Функция 14 INT 10h – выводит символ на экран и сдвигает курсор на следующую позицию. Во время вызова функции:

AL – выводимый символ

BH – номер страницы

```
MOV AH, 14
MOV AL, ‘A’
MOV BH, 0
INT  10h
```

Функции вывода на экран с помощью инструментов DOS:

Функция Int 21h 2 : Выводит символ на дисплей вместе со сдвиганием курсора. При вызове:

DL – символ

```
MOV AH, 2
MOV DL, ‘A’
INT  21h
```

Функция 9 Int 21h: Выводит строку символов на дисплей.

При вызове:

DS:DX – адрес строки символов заканчивающихся знаком $.

Курсор двигается в конец строки.

```
; в сегменте данных
STRING DB      ‘Текстовая строка $’
;сегмент кода
```

```
MOV AH, 9
LEA   DX, STRING
INT    21h
```

Функция 40h Int 21h: Запись файлов или вывод данных в устройство.

При вызове:

BX – логический номер файла или устройства. Логический номер дисплея – 1;

CX – количество выводимых символов;

DS:DX – адрес буфер откуда берутся данные.

При возврате:

Если CF=0, тогда AX – количество записанных байтов. Если CF=1, тогда в AX – код возврата (код ошибки).

Знак DS:DX, описываемый при каждом вызове любой функции, используется следующим образом. В сегменте данных, или в сегменте кода, строка символов загружаемых в регистр DX должна быть описана. Это выполняется с помощью следующих команд:

```
LEA   DX, <имя строки символов>          ;или
MOV DX, offset <имя строки символов>
```

Адрес сегмента, где расположена строка символов содержится в регистре DS.

```
;в сегменте данных
STRING          DB  ‘текстовая строка’
STRINGLEN EQU     $-STRING
;в сегменте кода
MOV AH, 40h
MOV BX, 1              ;логический номер дисплея
MOV CX, STRINGLEN
LEA   DX, STRING
INT    21h
```

1-пример. Вывести горизонтальную линию.

```
MOV CX, 10
M1:
MOV AH, 14
MOV AL, 196          ; ASCII код символа ‘-‘
```

```
MOV BH, 0
INT 10H
LOOP M1
```

2-пример. Вывести горизонтальную линию со сменой атрибута. Жёлтые символы на синем фоне.

```
MOV AH, 9
MOV BH, 0          ;текущая страница
MOV CX, 15
MOV AL, '='        ;выводимый символ
MOV BL, 00011110B
INT 10h
```

Описание данных.

Переменные в сегменте данных можно описать с помощью псевдооператоров DB, DW, DD.

Например,

```
A   DB  1, 2, 3, 4
B   DW  1122h, 3344h
C   DD  11223344h, 55667788h
```

Если нужно работать с байтами, на них можно ссылаться с помощью оперftора byte ptr.

```
MOV AL, A              ;AL=1
MOV AL, byte ptr B     ;AL=22h
MOV AL, byte ptr C     ;AL=44h
```

Во втором и третьем примере в регистр AL записываются младшие байты слов и двойных слов.

Чтобы сослаться на старшие байты нужно добавить адресу необходимое смещение:

```
MOV AL, byte ptr B  +1; AL=11h
MOV AL, byte ptr C  +3; AL=11h
```

Если нужно работать со словами, ссылка на них осуществляется с помощью оператора word ptr.

```
MOV AX,  word ptr A;AX=0201h
MOV AX,  B                  ;AX=1122h
MOV AX,  word ptr C;AX=3344h
```

Для ссылки на большие слова к адресу добавляется смещение для слов:

```
MOV AX,  word ptr C+2     ;AX=1122h
```

В программах используются постоянные (константы) описываемые с помощью директив EQU или «=».

```
;сегмент данных
K       EQU 255
A       DB  ?
;сегмент кода
MOV A,    K
```

В этом случае второй операнд становится непосредственным, первый операнд показывает адрес.

Массивы можно задать в следующем виде:

```
MAS_A           DB  1,2,3,4,5,6,7,8,9
MAS_B           DB  31H, 32H, ..., 39H
MAS_C           DB  '1,2,3,4,5,6,7,8,9'  ; Этот массив в памяти занимает 17 байт обьёма, по той причине что содержит запятые.
MAS_D           DB  1B, 10B, 11B,  100B, 101B, 110B, 111B, 1000B
;массив задан в двоичном коде.
```

Перед выводом на экран массивов заданных в двоичном и десятичном виде нужно преобразовать в ASCII код.

```
MOV AH, 2
MOV DL, mas[SI]
ADD  DL,30H; взят ASCII код
INT   21H
```

Символьные строки передаются в ASCII кодировке. Поэтому в обязательном порядке помещается в кавычки.

```
StringA     DB  'При (использовании 9 функции)'
            DB  'INT 21H текстовая строка', 13, 10
            DB  'должна заканчиваться символом $ (доллар)'
StringB     DB  'При приёме функции INT 21H 40h'
            DB  'конец текстовой строки', 13, 10
            DB  'определяется счётчиком CX'
```

Во время вывода рамки на дисплей его можно описать в сегменте данных в виде текстовой строки.

Во время вывода символа на дисплей в регистр DL (функция 2 INT 21H) загружается ASCII код символа:

```
MOV DL, 41h  ;ASCII код символа ’A’
```

или

```
MOV DL, ‘A’
```

3-пример.

```
SSEG        SEGMENT
        DB 128 DUP(?)
SSEG ENDS
DSEGSEGMENT
STRING DB       ‘Текстовая строка’
STRINGLEN EQU  $-STRING
DSEG        ENDS
CSEGSEGMENT
ASSUME   DS:DSEG,  CS:CSEG,  SS:SSEG
START:
        MOV        AX, DSEG
        MOV  DS, AX
; Вывод символьной строки на экран
        MOV        AH, 40H
        MOV        BX, 1                ;логический номер дисплея
        MOV        CX, STRINGLEN
        LEA DX, STRING
        INT  21H
        ;Окончание программы
        MOV        AH,  4CH
        INT  21h
 CSEG       ENDS
 END START
```

Обзор системных функции работы с файлами.

Все эти функции относятся к расширенной версии DOS, в них обращение к файлу осуществляется через дескриптор. Дескриптор или логический номер файла, присваивается операционной системой файлу в

результате создания или открытия файла, заданного в формате ASCIIZ-строки. Строка ASCIIZ – символьная строка, она заканчивается нулём. Дескриптор файла записывается в регистр AX. Его нужно сохранить в ячейку памяти.

Первые пять номеров (0-4) присвоено стандартным устройствам:

0 – стандартный ввод (CON);

1 – стандартный вывод (CON);

2 – стандартная ошибка (CON);

3 – стандартный (вспомогательный) порт (AUX);

4 – стандартный принтер (PRN).

Мы можем использовать дескриптор 0 для ввода с клавиатуры, и дескрипторы 1 и 2 для вывода на экран.

Во время работы с файлами удачно выполненная операция записывает 0 во флаг CF. Если операция не выполняется , тогда во флаг CF записывается 1, в регистр AX помещается код ошибки.

Коды ошибки

Код ошибки	Описание	Код ошибки	Описание
01	ошибка номера функции	10	ошибка оборудования
02	файл не найден	11	ошибка формата
03	путь доступа не найден	12	ошибка кода доступа
04	открыто слишком много файлов	13	ошибка данных
05	доступ не разрешён	15	ошибка дисковода
06	ошибка файлового номера	16	попытка удалить оглавление
07	блок управления памятью разрушен	17	не то устройство
08	не достаточно памяти	18	нет больше файлов
09	ошибка адреса		

Функции 3CH и 5BH дают возможность создать файл с заданной спецификацией (путь и имя файла – в формате строки ASCIIZ). Различие функций в том, что функция 3Ch уничтожает имеющийся файл и создает

новый с тем же именем, а функция 5Bh завершается с CF=1, если файл с таким именем существует.

Пример:

```
; в  сегмент  данных поместить имя файла и зарезервировать ячейку для дескриптора
   fname   DB   'F1.TXT',0
   handle  DW  ?
      . . .
      mov  AH, 3Ch          ; запрос на создание
      mov  CX, 0            ; обычного файла
      lea    DX, fname      ; адрес имени файла
      int    21h
      jc     err            ; переход по ошибке
      mov  handle, AX       ; сохранить дескриптор
```

В этом примере в регистр CX заносится атрибут файла. Могут использоваться следующие атрибуты:

01h - файл только для чтения

02h - скрытый файл

04h - системный файл

Функция 3Dh - позволяет открыть уже имеющийся файл.

Пример:

```
   mov  AH, 3Dh          ; функция открытия файла
   mov  AL, 2            ; для ввода-вывода
   lea    DX, fname
   int    21h
   mov  handle, AX
```

При открытии файла в регистр AL заносится код доступа:

0 - открыть файл только для ввода;

1 - открыть файл только для вывода;

2 - открыть файл для ввода и вывода.

Функция 42h используется для организации прямого доступа к произвольному месту файла. Указатель можно установить на начало файла (AL=0), в текущее положение (AL=1) и в конце файла (AL=2).

Кроме того, значение смещения указателя заносится в регистр CX (старшая половина) и DX (младшая половина).

Пример:

```
mov  BX, handle
mov  AH, 42h          ; установить указатель
mov  AL, 2            ; на конец файла
mov  CX, 0
mov  DX, 0
int  21h
```

Функция 3Fh используется для чтения из файла или устройства. При попытке чтения за концом файла AX=0.

Пример:

```
mov  AH, 3Fh          ; функция чтения
mov  BX, handle       ; установить дескриптор
mov  CX, 80           ; сколько читать
lea  DX, buf          ; и куда
int  21h
mov  CX, AX           ; сколько фактически прочитали
```

Функция 40h используется для записи в файл или на устройство.

Пример:

```
mov  AH, 40h          ; функция записи
mov  BX, handle       ; установка дескриптора
mov  CX, 80           ; сколько писать
lea  DX, buf          ; откуда
int  21h
```

Функция 43h используется для получения (AL=0) или установки (AL=1) атрибутов файла.

Пример:

```
mov  AH, 43h          ; функция работы с атрибутами
mov  AL, 1            ; установка атрибутов
mov  CX, 1            ; "только для чтения"
mov  DX, offset fname ; адрес имени файла
int  21h
```

Функция 56h используется для переименования файла.

Пример:

```
; настроить сегментный регистр ES на сегмент данных
   push DS
   pop  ES
 ; переименовать файл
   mov  AH, 56h                ; функция переименования
   mov  DX, offset oldname     ; адрес старого имени
   mov  DI,  offest newname    ; адрес нового имени
   int   21h
; в сегмент данных
oldname      db    'fl.txt',0
newname      db    'newf.txt',0
```

Функция 57h используется для получения (AL=0) и установки (AL=1) даты и времени создания файла. Время записывается в регистр CX и вычисляется по формуле:

CX= часы*2048+ минуты*32+секунды/2

Дата записывается в регистр DX и вычисляется по формуле:

DX=(год-1980)*512+месяц*32+день.

Пример:

```
; изменить время и дату создания файла
   mov  AH, 57h          ; функция даты/времени
   mov  AL, 1            ; установить дату/время
   mov  BX, handle       ; установить дескриптор
   mov  CX, 0            ; очистить cx
   or    CX, sec         ; добавить секунды
   or    CX, min         ; добавить минуты
   or    CX, hour        ; добавить часы
   xor   DX, DX          ; очистить DX
   or    DX,day          ; добавить день
   or    DX, mon         ; добавить месяц
   or    DX, year        ; добавить год
   int   21h
; в сегмент данных
sec             DW  6/2               ; 6 секунд
```

```
min     DW  15 * 32          ; 15 минут
hour    DW  16*2048          ; 16 часов
day         DW  25           ; 25 число
mon     DW  3*32             ; Март
year    DW  25*512           ; 25 лет от 1980г, т.е. 2005 г.
```

Функция 3Eh используется для закрытия файла. Эта операция необходима для коррекции оглавления и таблицы FAT.

Пример:

```
mov  AH, 3Eh        ; функция закрытия
mov  BX, handle
int  21h
```

Функция 41h используется для удаления файлов (за исключением файлов с атрибутом 'только чтение')

```
mov  AH, 41h        ;  функция удаления
lea  DX, fname      ;  адрес имени файла
int  21h            ;  вызов DOS
```

Работа с файлами может состоять из нескольких этапов:

1) Создания и открытие файла;

2) Установить указатель;

3) Определение длину файла;

4) Чтение нужного байта, блока, запись байта, блока, дополнение файла;

5) Закрытие файла;

6) Удаление файла.

При разработке программы не все этапы необходимы, а также они могут следовать в другом порядке. При чтении нужного байта можно указать либо номер байта, либо значение.

4-пример. Прочитать 10-й байт, для этого открыть файл для чтения/записи

```
...
;  установить указатель на 10 байт
    mov  AH, 42h            ; установить указатель
    mov  BX, handle
    mov  AL, 0              ; на начало файла
    mov  CX, 0
    mov  DX, 10             ; на 10-й байт
```

```
        int     21h
; Прочитать нужный байт
        mov  ah, 3Fh
        mov  bx, HANDLE
        lea     dx, BUF
        mov  cx, 1
        int     21h
; в сегмент данных

     BUF    DB   ?
```

5-пример. Найти символ "a" и определить его номер.

```
    ;открыть файл
        ....
    ;определить длину файла
        mov  AH, 42h                ; установить указатель
        mov  BX, HANDLE
        mov  AL, 2                  ; на конец файла
        mov  CX, 0                  ; старшая половина смещения
        mov  DX, 0                  ; младшая половина смещения
        int     21h;
        mov  FLEN, AX               ; запомнить длину файла в ячейке FLEN
    ;установить указатель на начало файла
         ....
    ;прочитать файл в bufin
        ....
    ;поиск символа "a"
        cld                 ; поиск осуществляется слева направо
        mov  CX, FLEN       ; длина файла
        mov  AL, 'A'        ; искомый символ
        lea     DI, BUFIN   ; адрес строки
        repne       scasb   ; сканировать
        jnz     m           ; если символ не найден, перейтди в M
        dec     DI          ; символ найден - уменьшить адрес
        ...                 ; вывести номер на экран
M:      ...
    ; вывести сообщение, что символ не найден
```

; закрыть файл

Контрольные вопросы.

1. Чем отличаются функции 1, 7, 8 int 21h?
2. Что записывается в первый и во второй байт буфера при использовании функции 0Ah int 21h?
3. Как выполняется ввод при нажатии функциональной клавиши?
4. Чем отличается выполнение функции 0 (int 16h) и 7 (int 21h)?
5. Чем отличаются функции 0Ah и 3Fh int 21h?
6. С какого байта в буфере начинаются данные после использования функции 0Ah int 21h?
7. Чему равно значение последнего байта строки после использования функции 0Ah int 21h?
8. Функция вывода символа на экран.
9. Функция вывода строки на экран.
10. Как определяется конец строки при использовании функции 9 и 40h int 21h?
11. Какие функции не перемещают курсор при выводе символа?
12. Какая функция вывода на дисплей меняет атрибут символа?
13. Что такое дескриптор файла?
14. Что такое ASCIIZ строка?
15. Дескрипторы стандартного ввода и стандартного вывода.
16. Как определяется успешность завершения какой-либо файловой операции?
17. Атрибуты файла.
18. Коды доступа при открытии файла.
19. Как определить длину файла?
20. Что делает функция закрытия файла?
21. Какие файлы нельзя удалить с помощью функции 41h?

Лабораторная работа 6. Разработка графического приложения для Windows

Цель работы. Изучить основы программирования в среде Win32

Задания

1. Изучить процедуру главного окна
2. Изучить цикл обработки сообщений
3. Изучить используемые в программе API-функции
4. Изменить параметры функции GetMessageA
5. Изменить параметры функции CreateWindowExA
6. Изменить главную процедуру окна WNDPROC

Методические рекомендации по выполнению лабораторной работы

Рассмотрим простую программу графического приложения. В таблице MASM есть разные библиотеки. В нашем примере используются две: user32.lib kernel32.lib. Они добавляются с помощью директивы includelib.

Сначала определяются константы и внешние библиотечные процедуры. В действительности все эти определения можно найти в include-файлах, прилагаемых к пакету TASM32. Мы не будем использовать стандартные include-файлы, т.к. во-первых, так удобнее понять технологию программирования, во-вторых, будет легко с MASM на TASM.

Необходимо четко понимать способ функционирования процедуры окна, так как именно это определяет суть программирования под Windows. Задача данной процедуры – правильная реакция на все приходящие сообщения. Все необработанные сообщения должны возвращаться в систему при помощи функции DefWindowProcA. Мы рассмотрим четыре сообщения: WM_CREATE, WM_DESTROY, WM_LBUTTONDOWN, WM_RBUTTONDOWN.

Сообщения WM_CREATE и WM_DESTROY в терминах объектного программирования играют роль конструктора и деструктора: они приходят в функцию окна при создании окна и при уничтожении окна. Если щелкнуть по крестику в правом углу окна, то в функцию окна придет сообщение WM_DESTROY. Далее будет выполнена функция PostQuitMessage и приложению будет послано сообщение WM_QUIT, которое вызовет выход из цикла ожидания и выполнение функции ExitProcess, что, в свою очередь, приведет к удалению приложения из памяти.

Обратите внимание на метку _ERR – переход на нее происходит при возникновении ошибки, и здесь можно поместить соответствующее сообщение.

```
.386P
.MODEL FLAT, stdcall
; константы
; сообщение приходит при закрытии окна
WM_DESTROY       equ   2
; сообщение приходит при создании окна
WM_CREATE        equ   1
; сообщение при щелчке левой кнопкой мыши в области окна
WM_LBUTTONDOWN   equ   201h
; сообщение при щелчке правой кнопкой мыши в области окна
WM_RBUTTONDOWN   equ   204h
; свойства окна
CS_VREDRAW       equ   1h
CS_HREDRAW       equ   2h
CS_GLOBALCLASS       equ   4000h
WS_OVERLAPPEDWINDOW equ   000CF0000H
style equ   CS_HREDRAW + CS_VREDRAW + CS_GLOBALCLASS
; идентификатор стандартной иконки
IDI_APPLICATION equ   32512
; идентификатор курсора
IDC_CROSS    equ   32515
; режим показа окна - нормальный
SW_SHOWNORMAL            equ   1
; прототипы внешних процедур
EXTERN  MessageBoxA: NEAR
EXTERN  CreateWindowExA: NEAR
EXTERN  DefWindowProcA: NEAR
EXTERN  DispatchMessageA: NEAR
EXTERN  ExitProcess: NEAR
EXTERN  GetMessageA: NEAR
EXTERN  GetModuleHandleA: NEAR
EXTERN   LoadCursorA: NEAR
EXTERN   LoadIconA: NEAR
EXTERN   PostQuitMessage: NEAR
EXTERN   RegisterClassA: NEAR
EXTERN   ShowWindow: NEAR
EXTERN   TranslateMessage: NEAR
EXTERN   UpdateWindow: NEAR
; директивы компоновщику для подключения библиотек
```

```
includelib c:\win32\tasm32\lib\import32.lib
;-------------------------------------------------------------------
; структуры
; структура сообщения
MSGSTRUCT STRUC
MSHWND DD  ?          ; идентификатор сообщения
MSWPARAM   DD  ?      ; доп. информация о сообщении
MSLPARAM   DD  ?      ; доп. информация о сообщении
MSTIME  DD  ?         ; время посылки сообщения
MSPT    DD  ?         ; положение курсора во время посылки
сообщения
MSGSTRUCT   ENDS
;-------------------------
WNDCLASS STRUC
CLSSTYLE          DD  ?     ; стиль окна
CLWNDPROC         DD  ?     ; указатель на процедуру окна
CLSCEXTRA         DD  ?     ; информация о доп. байтах для
данной структуры
CLWNDEXTRA        DD  ?     ; информация о доп. байтах для окна
CLSHINSTANCE      DD  ?     ; дескриптор приложения
CLSHICON          DD  ?     ; идентификатор иконки окна
CLSHCURSOR        DD  ?     ; идентификатор курсора окна
CLBKGROUND        DD  ?     ; идентификатор кисти окна
CLMENUNAME        DD  ?     ; имя-идентификатор меню
CLNAME       DD  ?          ; специфицирует имя класса окна
 WNDCLASS   ENDS

 ; сегмент данных
 _DATA SEGMENT DWORD PUBLIC USE32 'DATA'
      NEWHWND   DD  0
      MSGMSGSTRUCT  <?>
      WC  WNDCLASS  <?>
      HINST         DD  0     ; здесь хранится дескриптор
приложения
      TITLENAME   DB  'Простой пример 32-битного приложения',
0
      CLASSNAME  DB  'CLASS32', 0
      CAP       DB  'Сообщение', 0
      MES1          DB  'Вы нажали левую кнопку мыши', 0
      MES2          DB  'Выход из программы. Пока!', 0
```

```
_DATA    ENDS
; сегмент кода
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
START:
; получить дескриптор приложения
        PUSH    0
        CALL    GetModuleHandleA
        MOV     [HINST],EAX
REG_CLASS:
; заполнить структуру окна
        MOV     [WC.CLSSTYLE], style
; процедура обработки сообщений
        MOV     [WC.CLWNDPROC], OFFSET WNDPROC
        MOV     [WC.CLSCEXTRA], 0
        MOV     [WC.CLWNDEXTRA], 0
        MOV     EAX, [HINST]
        MOV     [WC.CLSHINSTANCE],EAX
;-----------------иконка окна
        PUSH    IDI_APPLICATION
        PUSH    0
        CALL    LoadIconA
        MOV     [WC.CLSHICON],    EAX
;-----------------курсор окна
        PUSH    IDC_CROSS
        PUSH    0
        CALL    LoadCursorA
        MOV     [WC.CLSHCURSOR], EAX
;------------------
        MOV     [WC.CLBKGROUND],17 ; цвет окна
        MOV     DWORD PTR [WC.CLMENUNAME], 0
        MOV     DWORD PTR [WC.CLNAME], OFFSET
CLASSNAME
        PUSH    OFFSET   WC
        CALL    RegisterClassA
; создать окно зарегистрированного класса
        PUSH    0
        PUSH    [HINST]
        PUSH    0
        PUSH    0
        PUSH    400         ; DY - высота окна
```

```
        PUSH    400         ; DX - ширина окна
        PUSH    100         ; Y - координата левого верхнего угла
        PUSH    100         ; X - координата левого верхнего угла
        PUSH    WS_OVERLAPPEDWINDOW
        PUSH    OFFSET  TITLENAME   ; имя окна
        PUSH    OFFSET  CLASSNAME   ; имя класса
        PUSH    0
        CALL    CreateWindowExA
    ; проверка на ошибку
        CMPEAX, 0
        JZ   _ERR
        MOV     [NEWHWND], EAX
        PUSH    SW_SHOWNORMAL
        PUSH    [NEWHWND]
        CALL    ShowWindow          ; показать созданное окно
        PUSH    [NEWHWND]
        CALL    UpdateWindow        ; команда перерисовать
видимую часть окна, сообщение WM_PAINT
    ; цикл обработки сообщений
    MSG_LOOP:
        PUSH    0
        PUSH    0
        PUSH    0
        PUSH    OFFSET  MSG
        CALL    GetMessageA
        CMPEAX, 0
        JE   END_LOOP
        PUSH    OFFSET  MSG
        CALL    TranslateMessage
        PUSH    OFFSET  MSG
        CALL    DispatchMessageA
        JMP MSG_LOOP
    END_LOOP:
    ; выход из программы (закрыть процесс)
        PUSH    [MSG.MSWPARAM]
        CALL    ExitProcess
    _ERR:
        JMP END_LOOP
    ;---------------------------------------------------------------------------
    ; процедура окна
```

```
; расположение параметров в стеке
; [EBP+14H]  LPARAM
; [EBP+10H]  WPARAM
; [EBP+0CH]  MES
; [EBP+8]    HWND
WNDPROC    PROC
        PUSH    EBP
        MOV     EBP, ESP
        PUSH    EBX
        PUSH    ESI
        PUSH    EDI
        CMP DWORD PTR [EBP+0CH], WM_DESTROY
        JE  WMDESTROY
        CMP DWORD PTR [EBP+0CH], WM_CREATE
        JE  WMCREATE
        CMP DWORD PTR [EBP+0CH], WM_LBUTTONDOWN  ; левая
кнопка мыши
        JE  LBUTTON
        CMP DWORD PTR [EBP+0CH], WM_RBUTTONDOWN  ;
правая кнопка мыши
        JE  RBUTTON
        JMP DEFWNDPROC
; нажатие правой кнопки мыши приводит к закрытию окна
RBUTTON:
        JMP WMDESTROY
; нажатие левой кнопки мыши
LBUTTON:
; выводим сообщение
        PUSH    0           ; MB_OK
        PUSH    OFFSET CAP
        PUSH    OFFSET  MES1
        PUSH    DWORD PTR [EBP+08H]
        CALL    MessageBoxA
        MOV     EAX, 0
        JMP FINISH
WMCREATE:
        MOV     EAX, 0
        JMP FINISH
DEFWNDPROC:
        PUSH    DWORD PTR [EBP+14H]
```

```
        PUSH    DWORD PTR [EBP+10H]
        PUSH    DWORD PTR [EBP+0CH]
        PUSH    DWORD PTR [EBP+08H]
        CALL    DefWindowProcA
        JMP FINISH
    WMDESTROY:
        PUSH    0                           ; MB_OK
        PUSH    OFFSET  CAP
        PUSH    OFFSET  MES2
        PUSH    DWORD PTR [EBP+08H] ; дескриптор окна
        CALL    MessageBoxA
        PUSH    0
        CALL    PostQuitMessage             ; сообщение WM_QUIT
        MOV     EAX, 0
    FINISH:
        POP EDI
        POP ESI
        POP EBX
        POP EBP
        RET 16
    WNDPROC     ENDP
    _TEXT   ENDS
    END START
```

Контрольные вопросы.

1. Какие API-функции используются в программе?
2. Какие структуры используются в программе?
3. Структура графического приложения.
4. Какие API-функции включает цикл обработки сообщений?

Лабораторная работа №7. Консольные приложения

Цель работы. Изучение основ применения API функции для консольных приложений.

Задание.

1. Изменить размер окна консоли.
2. Поменять заголовок окна консоли.
3. Изменить позиции курсора.
4. Поменять цветовые атрибуты текста.

5. Разобрать алгоритм работы процедур NUMPAR и GETPAR.

Методические рекомендации по выполнению лабораторной работы

Для создания консоли используем функцию AllocConsole. По завершении программы консоль освобождается автоматически или с помощью функций FreeConsole.

Следует отметить, что один процесс может иметь только одну консоль, поэтому выполнение функции FreeConsole в начале программы обязательно.

Для чтения из буфера консоли используется функция ReadConsole.

Установить позицию курсора в консоли можно при помощи функции SetConsoleCursorPosition.

Установить цвет выводимых букв можно с помощью функции SetConsoleTextAttribute.

Для определения заголовка окна консоли используется функция SetConsoleTitle.

Функция CharToOem используется для перевода DOS-кодировки в MS-DOS-кодировку.

Большинство консольных функций при правильном их завершении возвращает ненулевое значение. В случае ошибки в EAX помещается ноль.

Пример-1. Создание собственной консоли

```
; cons1.asm
.386P
.MODEL FLAT, stdcall
; константы
STD_OUTPUT_HANDLE equ -11
STD_INPUT_HANDLE  equ -10
; атрибуты цветов
FOREGROUND_BLUE    equ 1h    ; синий цвет букв
FOREGROUND_GREEN   equ 2h    ; зеленый цвет букв
FOREGROUND_RED     equ 4h    ; красный цвет букв
FOREGROUND_INTENSITY equ 8h  ; повышенная
интенсивность
BACKGROUND_BLUE    equ 10h       ; синий цвет фона
BACKGROUND_GREEN   equ 20h       ; зеленый цвет фона
BACKGROUND_RED     equ 40h   ; красный цвет фона
BACKGROUND_INTENSITY equ 80h ; повышенная
интенсивность
COL1 = 2h + 8h; цвет выводимого текста
```

```
COL2 = 1h + 2h + 8h ; цвет выводимого текста 2
; прототипы внешних процедур
EXTERN  GetStdHandle: NEAR
EXTERN  WriteConsoleA: NEAR
EXTERN  SetConsoleCursorPosition: NEAR
EXTERN  SetConsoleTitleA: NEAR
EXTERN  FreeConsole: NEAR
EXTERN  AllocConsole: NEAR
EXTERN  CharToOemA: NEAR
EXTERN  SetConsoleCursorPosition: NEAR
EXTERN  SetConsoleTextAttribute: NEAR
EXTERN  ReadConsoleA: NEAR
EXTERN  SetConsoleScreenBufferSize: NEAR
EXTERN  ExitProcess: NEAR
; директивы компоновщику для подключения библиотек
includelib c:\win32\work\import32.lib
;-------------------------------------------------------------------------
COOR    STRUC
        X       WORD            ?
        Y       WORD            ?
COOR    ENDS
; сегмент данных
_DATA SEGMENT DWORD PUBLIC USE32 'DATA'
            HANDL   DWORD   ?
            HANDL1  DWORD   ?
            STR1            DB   'Введите строку: ', 13, 10, 0
STR2            DB   'Простой пример работы консоли', 0
BUF         DB   200 dup (?)
LENS            DD   ?      ; количество выведенных символов
            CRD         COOR    <?>
_DATA ENDS

_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
START:
; перекодируем строку
        PUSH        OFFSET   STR1
            PUSH        OFFSET   STR1
        CALL        CharToOemA
; образовать консоль
; вначале освободить уже существующую
```

```
        CALL    FreeConsole
        CALL    AllocConsole
; получить HANDL1 ввода
                PUSH    STD_INPUT_HANDLE
                CALL    GetStdHandle
                MOV     HANDL1, EAX
; получить HANDL вывода
                PUSH    STD_OUTPUT_HANDLE
                CALL    GetStdHandle
                MOV     HANDL, EAX
; установить новый размер окна консоли
                MOV     CRD.X, 100
                MOV     CRD.Y, 25
                PUSH    CRD
                PUSH    EAX
        CALL    SetConsoleScreenBufferSize
; задать заголовок консоли
                PUSH    OFFSET   STR2
        CALL    SetConsoleTitleA
; установить позицию курсора
                MOV     CRD.X, 0
                MOV     CRD.Y, 10
        PUSH    CRD
                PUSH    HANDL
        CALL    SetConsoleCursorPosition
; задать цветовые атрибуты выводимого  текста
        PUSH    COL1
                PUSH    HANDL
                CALL    SetConsoleTextAttribute
; вывести строку
        PUSH    OFFSET    STR1
        CALL    LENSTR  ; в EBX длина строки
        PUSH    0
        PUSH    OFFSET   LENS
                PUSH    EBX
        PUSH    OFFSET   STR1
                PUSH    HANDL
        CALL    WriteConsoleA
; ждать ввода строки
        PUSH    0
```

```
            PUSH    OFFSET  LENS
            PUSH    200
            PUSH    OFFSET  BUF
            PUSH    HANDL1
            CALL    ReadConsoleA
; вывести полученную строку
; вначале задать цветовые атрибуты выводимого текста
            PUSH    COL2
            PUSH    HANDL
            CALL    SetConsoleTextAttribute
;-------------------------------------------------
      PUSH    0
            PUSH    OFFSET  LENS
            PUSH    [LENS]
            PUSH    OFFSET  BUF
      PUSH    HANDL
      CALL    WriteConsoleA
; небольшая задержка
      MOV     ECX, 01FFFFFFFh
L1:
            LOOP    L1
; закрыть консоль
      CALL    FreeConsole
      CALL    ExitProcess
; строка - [EBP+08h]
; длина в EBX
LENSTR PROC
;     PUSH    EBX
;     MOV     EBP, ESP
            ENTER   0, 0
            PUSH    EAX
;-----------------------------------
            CLD
            MOV     EDI, DWORD PTR [EBP+08h]
            MOV     EBX, EDI
            MOV     ECX, 100  ; ограничить длину строки
            XOR AL, AL
            REPNE     SCASB   ; найти символ 0
            SUB EDI, EBX ; длина строки, включая 0
            MOV     EBX, EDI
```

```
        DEC EBX
;       POP EAX
;       POP EBP
        LEAVE
            RET 4
LENSTR  ENDP
_TEXT ENDS
END START
```

Для работы с командной строкой применяется API функция GetCommandLineA, которая возвращает указатель на командную строку . Эта функция одинаково работает как для консольных приложений, так и для приложений GUI (Graphic Universal Interface).

Пример-2. Программа вывода параметров командной строки

```
; cons_2.asm
.386P
.MODEL FLAT, stdcall
; константы
STD_OUTPUT_HANDLE equ   -11
; прототипы внешних процедур
EXTERN  GetStdHandle: NEAR
EXTERN  WriteConsoleA: NEAR
EXTERN  ExitProcess: NEAR
EXTERN  GetCommandLineA: NEAR
; директивы компоновщику для подключения библиотек
includelib c:\win32\work\import32.lib
; сегмент данных
_DATA SEGMENT DWORD PUBLIC USE32 'DATA'
        HANDL   DWORD   ?
        NUM             DWORD   ?
        BUF     DB      100 dup (0)
        LENS            DWORD   ?       ; количество выведенных
символов
        CNT     DWORD   ?
_DATA ENDS
; сегмент кода
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
START:
; получить HANDLE вывода
        PUSH    STD_OUTPUT_HANDLE
        CALL    GetStdHandle
```

```
        MOV     HANDL, EAX
; получить количество параметров
        CALL    NUMPAR
        MOV     NUM, EAX
        MOV     CNT, 0
;-----------------------------------------
; вывести параметры командной строки
LL1:
        MOV     EDI, CNT
        CMPNUM, EDI
        JE   LL2        ; номер параметра
        INC  EDI
        MOV     CNT, EDI
; получить параметр номером EDI
        LEA EBX, BUF
        CALL    GETPAR
; получить длину параметра
        PUSH    OFFSET BUF
        CALL    LENSTR
; в конце - перевод строки
        MOV     BYTE PTR [BUF+EBX], 13
        MOV     BYTE PTR [BUF+EBX+1], 10
        MOV     BYTE PTR [BUF+EBX+2], 0
        ADDEBX, 2
; вывод строки
        PUSH    0
        PUSH    OFFSET LENS
        PUSH    EBX
        PUSH    OFFSET BUF
        PUSH    HANDL
        CALL    WriteConsoleA
        JMP LL1
LL2:
        PUSH    0
        CALL    ExitProcess
; строка - [EBP+08H]
; длина в EBX
LENSTR PROC
        PUSH    EBX
        MOV     EBP, ESP
```

```
        PUSH    EAX
;---------------------------------------
        CLD
        MOV     EDI, DWORD PTR [EBP+08H]
        MOV     EBX, EDI
        MOV     ECX, 100        ; ограничить длину строки
        XOR AL, AL
        REPNE    SCASB          ; найти символ 0
        SUB EDI, EBX        ; длина строки, включая 0
        MOV     EBX, EDI
        DEC EBX
;---------------------------------------
        POP EAX
        POP EBP
        RET 4
LENSTR  ENDP
; определить количество параметров (->EAX)
NUMPAR      PROC
        LOCALS
        CALL    GetCommandLineA
        MOV     ESI, EAX        ; указатель на строку
        XOR ECX, ECX            ; счетчик
        MOV     EDX, 1
@@L1:
        CMP BYTE PTR [ESI], 0
        JE  @@L4
        CMP BYTE PTR [ESI], 32
        JE  @@L3
        ADD ECX, EDX            ; номер параметра
        MOV     EDX, 0
        JMP @@L2
@@L3:
        OR  EDX, 1
@@L2:
        INC ESI
        JMP @@L1
@@L4:
        MOV     EAX, ECX
        RET
NUMPAR      ENDP            ; получить параметр
```

```
; EBX - указывает на буфер, куда будет помещен параметр
; в буфер помещается строка с нулем на конце
; EDI - номер параметра
GETPAR  PROC
        LOCALS
        CALL    GetCommandLineA
        MOV     ESI, EAX        ; указатель на строку
        XOR ECX, ECX            ; счетчик
        MOV     EDI, 1
@@L1:
        CMP BYTE PTR [ESI], 0
        JE   @@L4
        CMP BYTE PTR [ESI], 32
        JE   @@L3
        ADD ECX, EDX            ; номер параметра
        MOV     EDX, 0
        JMP @@L2
@@L3:
        OR   EDX, 1
@@L2:
        CMP ECX, EDI
        JNE  @@L5
        MOV     AL, BYTE PTR [ESI]
        MOV     BYTE PTR [EBX], AL
        INC  EBX
@@L5:
        INC  ESI
        JMP @@L1
@@L4:
        MOV     BYTE PTR [EBX], 0
        RET
GETPAR  ENDP
_TEXT ENDS
END START
```

Для компиляции программы используются следующие командные строки:

```
TASM32 /ml prog.asm
TLINK32 /ap pgog.obj
```

Контрольные вопросы.

1. Какие аргументы использует API функция GetStdHandle?
2. Дайте названия цветов букв и символов используемых функцией SetConsoleTextAttribute?
3. Назовите параметры функции CharToOem?
4. Какие типы событий зарезервированы операционной системой?

Список литературы

1. Юров В., Хорошенко С. Ассемблер. – СПб.: «Питер», 2009. -470 с.
2. Зубков С. В. Assembler для DOS, Windows и UNIX. – М.: ДМК, 2014. – 378с.
3. Пирогов В. Ю. Assembler. Учебный курс. – М.: «Нолидж», 2012. – 589с.
4. Финогенов К. Г. Самоучитель по системным функциям MS-DOS. – М.: МП «МАЛИП», 2008. – 432с.
5. Пирогов В. Ю. Assembler для Windows. – М.: Молгачева С. В., 2013. – 521с.
6. Том Сван. Освоение Turbo Assembler. – М.: «Диалектика», 2006. – 312с.
7. Рудаков П.И., Финогенов К. Г. Язык Ассемблера: уроки программирования. – М.: «ДИАЛОГ-МИФИ», 2005. – 542с.
8. Скэнлон Л. ПЭВМ IBM PC и XT. Программирование на языке ассемблера. – М.: «Радио и Связь», 2007. – 368с.
9. Абель П. Язык Ассемблера для IBM PC и программирование. – М.: «Высшая школа», 2014. – 687с.
10. Брэдли Л. Программирование на языке Ассемблера для персональных ЭВМ IBM. – М.: «Радио и Связь», 2005. – 354с.
11. Нортон П., Соухэ Д. Язык Ассемблера для IBM PC: Пер. с англ., - М.: «Компьютер», 2013. – 741с.
12. Нортон П. ПК фирмы IBM и OS MS-DOS. –М.: «Радио и Связь», 2006. – 365с.
13. Пузанков Д. В. Микропорцессорные системы. – СПб.: «Политехника», 2007, - 935с.
14. Система программирования на макроассемблере MS-DOS (в 4-х частях). – Киев: НПО «Горсистемотехника», 2006. – 874с.

Содержание