МИНОБРНАУКИ РОССИИ

Государственное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им.В.И.Ульянова (Ленина)» (СПбГЭТУ)

Кафедра математического обеспечение и применения ЭВМ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ
по дисциплине

Организация ЭВМ и систем

Автор к.т.н., доцент В.А.Кирьянчиков

Санкт-Петербург
2018

Содержание

Лабораторная работа 1.
Трансляции, отладка и выполнение программ на языке Ассемблера.

В каталоге Comput_Org/Лабораторные работы/ содержатся 2 подкаталога: 1) MASM (с транслятором masm, компоновщиком link, отладчиком afd и библиотекой lib) и 2) Задания, в котором приведены описания заданий 9 лабораторных работ. Обязательными для получения зачета являются первые 6 работ, выполнение любой из работ 7 – 9 (по выбору студента) позволяет получить бонусы к экзамену (0.5 балла за 7 работ, 1 балл за 8 работ). Все работы на «старых» ПК (≤ 32-бит и ОС Windows ≤ XP) выполняются из FAR с запуском в командной строке, так как используют режим DOS.
При работе на «новых» ПК (≥ 64-бит и ОС Windows > XP) необходимо работать под эмулятором DOSBOX (для личной машины скачать из Интернета). Инструкция по работе с DOSBOX приведена в приложении 1 к данной работе.

Лабораторная работа 1 использует 2 готовых программы на ассемблере: hello1 – составлена с использованием сокращенного описания сегментов и hello2 – составлена с полным описанием сегментов и выводом строки, оформленным как процедура. Выполнение работы состоит из двух частей, по каждой из которых необходимо представить протокол с фиксацией всех выполняемых действий и полученных результатов, и подписать его у преподавателя.
Уточнение задания следует посмотреть в файле lr1_comp.txt каталога Задания.

Часть 1

1. Просмотреть программу hello1.asm, которая формирует и выводит на экран приветствие пользователя с помощью функции ОС MSDOS, вызываемой через прерывание с номером 21H (команда Int 21h).
   Выполняемые функцией действия и задаваемые ей параметры - следующие:
   - обеспечивается вывод на экран строки символов, заканчивающейся знаком "$";
   - требуется задание в регистре ah номера функции, равного 09h, а в регистре dx - смещения адреса выводимой строки;
   - используется регистр ax и не сохраняется его содержимое.
2. Разобраться в структуре и реализации каждого сегмента программы. Непонятные фрагменты прояснить у преподавателя. Строку-приветствие преобразовать в соответствии со своими личными данными.
3. Загрузить файл hello1.asm из каталога Задания в каталог Masm.
4. Протранслировать программу с помощью строки

   > masm hello1.asm

с созданием объектного файла и файла диагностических сообщений (файла листинга). Объяснить и исправить синтаксические ошибки, если они будут обнаружены транслятором. Повторить трансляцию программы до получения объектного модуля.

5. Скомпоновать загрузочный модуль с помощью строки

   > link hello1.obj

с созданием карты памяти и исполняемого файла hello1.exe.

6. Выполнить программу в автоматическом режиме путем набора строки

   > hello1.exe

убедиться в корректности ее работы и зафиксировать результат выполнения в протоколе.

7. Запустить выполнение программы под управлением отладчика с помощью команды

   > afd hello1.exe

Записать начальное содержимое сегментных регистров CS, DS, ES и SS. Выполнить программу в пошаговом режиме с фиксацией используемых регистров и ячеек памяти до и после выполнения каждой команды. Обычные команды выполняются по F1 (Step), а вызовы обработчиков прерываний (Int) - по F2 (StepProc), чтобы не входить внутрь обработчика прерываний. Продвижение по сегментам экранной формы отладчика выполняется с помощью клавиш F7 – F10 (up, down, left, right). Перезапуск программы в отладчике выполняется клавишей F3 (Retrieve). Выход из отладчика - по команде Quit.

Результаты прогона программы под управлением отладчика должны быть представлены в виде, показанном на примере одной команды в табл.1, и подписаны преподавателем.

Табл.1

Адрес Команды	Символический код команды	16-ричный код команды	Содержимое регистров и ячеек памяти	
			до выполнения .	После выполнения
0003	Mov DS, AX	8E D8	(AX) = 2D87 (DS) = 2D75 (IP) = 0003	(AX) = 2D87 (DS) = 2D87 (IP) = 0005

Часть 2

Выполнить пункты 1 - 7 части 1 настоящего задания применительно к программе hello2.asm, приведенной в каталоге Задания, которая выводит на экран приветствие пользователя с помощью процедуры WriteMsg, а также использует полное определение сегментов. Сравнить результаты прогона под управлением отладчика программ hello1 и hello2 и объяснить различия в размещении сегментов.

Отчет по работе должен содержать:

1) текст задания;
2) тексты исходных файлов программ hello1 и hello2;
3) тексты файлов диагностических сообщений hello1.lst и hello2.lst;
4) протокол работы на компьютере, включающий основные действия по пунктам 1 - 6 и протоколы пошагового исполнения каждой из программ под управлением отладчика в виде таблицы 1 (черновики протоколов должны быть подписаны преподавателем).
5) выводы по работе.

Приложение 1

Работа с DOSBox

В случае 64-разрядного Windows запустить транслятор Masm.exe напрямую не получится, т. к. 16-разрядный код в 64-разрядной Windows более не поддерживается. На помощь приходит эмулятор MS-DOS, под названием DOSBox. Его можно свободно скачать из Интернета с официальной страницы http://www.dosbox.com/. Существуют версии под многие операционные системы, сейчас нас интересует версия под Windows. Следует заметить, что в лабораториях кафедры DOSBox установлен и при выполнении программ им можно пользоваться. На личной машине скачиваем и устанавливаем DOSBox-0.74, следуя инструкциям программы-установщика. После успешной установки запускаем DOSBox.

Далее нужно смонтировать каталог с MASM в эмулятор DOSBox. Делается это командой mount <диск в эмуляторе> <реальный путь к каталогу с MASM>. В рассматриваемом примере смонтируем наш каталог C:\MASM_EXE как диск d: эмулятора командой

mount d c:\masm_exe

и перейдём на только что смонтированный диск, подав команду (в эмуляторе) d:
Содержимым диска d: в эмуляторе будет содержимое каталога C:\MASM_EXE реального компьютера, т. е. можно непосредственно запускать ассемблер, компоновщик и отладчик для работы с заданной программой.
Ещё одно замечание, которое необходимо сделать, касается переключения на русскую раскладку в DOSBox (важно для тех программ, в которых используются русские буквы). Для того, чтобы можно было переключаться на русский язык, в DOSBox необходимо загрузить кодовую страницу с буквами кириллицы, подав команду

keyb ru 866

(переключение на кодировку CP866, использовавшуюся в MS-DOS для работы с буквами русского алфавита). После этого переключение на русский язык будет доступно с помощью сочетания клавиш ⟨лев. Alt⟩+⟨прав. Shift⟩. Обратный переход на латинскую раскладку осуществляется сочетанием клавиш ⟨лев. Alt⟩+⟨лев. Shift⟩.

Лабораторная работа 2.
Изучение режимов адресации и формирования исполнительного адреса.

Задание:

Лабораторная работа 2 предназначена для изучения режимов адресации, использует готовую программу lr2_comp.asm на Ассемблере, которая в автоматическом режиме выполняться не должна, так как не имеет самостоятельного функционального назначения, а только тестирует режимы адресации. Поэтому ее выполнение должно производиться под управлением отладчика в пошаговом режиме.

В программу введен ряд ошибок, которые необходимо объяснить в отчете по работе, а соответствующие команды закомментировать для прохождения трансляции.
Необходимо составить протокол выполнения программы в пошаговом режиме отладчика по типу таблицы 1 предыдущей лабораторной работы и подписать его у преподавателя.
На защите студенты должны уметь объяснить результат выполнения каждой команды с учетом используемого вида адресации. Результаты, полученные с помощью отладчика, не являются объяснением, а только должны подтверждать ваши объяснения.

Порядок выполнения работы.

1. Получить у преподавателя вариант набора значений исходных данных (массивов) vec1, vec2 и matr из файла lr2.dat, приведенного в каталоге Задания и занести свои данные вместо значений, указанных в приведенной ниже программе.
2. Протранслировать программу с созданием файла диагностических сообщений; объяснить обнаруженные ошибки и закомментировать соответствующие операторы в тексте программы.
3. Снова протранслировать программу и скомпоновать загрузочный модуль.
4. Выполнить программу в пошаговом режиме под управлением отладчика с фиксацией содержимого используемых регистров и ячеек памяти до и после выполнения команды.

5. Результаты прогона программы под управлением отладчика должны быть подписаны преподавателем и представлены в отчете.
Пример используемой программы приведен ниже.

```
; Программа изучения режимов адресации процессора  IntelX86

EOL  EQU  '$'
ind  EQU   2
n1   EQU  500
n2   EQU  -50

; Стек  программы
AStack    SEGMENT  STACK
        DW 12 DUP(?)
AStack    ENDS

; Данные программы
DATA      SEGMENT

;  Директивы описания данных
mem1      DW    0
mem2      DW    0
mem3      DW    0
vec1      DB    1,2,3,4,8,7,6,5
vec2      DB    -10,-20,10,20,-30,-40,30,40
matr      DB    1,2,3,4,-4,-3,-2,-1,5,6,7,8,-8,-7,-6,-5
DATA      ENDS

; Код программы
CODE      SEGMENT
        ASSUME CS:CODE, DS:DATA, SS:AStack

; Головная процедура
Main      PROC  FAR
        push  DS
        sub   AX,AX
        push  AX
        mov   AX,DATA
        mov   DS,AX

;  ПРОВЕРКА РЕЖИМОВ АДРЕСАЦИИ НА УРОВНЕ СМЕЩЕНИЙ
;  Регистровая адресация
        mov  ax,n1
        mov  cx,ax
        mov  bl,EOL
        mov  bh,n2
;  Прямая   адресация
        mov  mem2,n2
        mov  bx,OFFSET vec1
        mov  mem1,ax
;  Косвенная адресация
        mov  al,[bx]
        mov  mem3,[bx]
;  Базированная адресация
```

```
        mov  al,[bx]+3
        mov  cx,3[bx]

;  Индексная адресация
        mov  di,ind
        mov  al,vec2[di]
        mov  cx,vec2[di]
;  Адресация с базированием и индексированием
        mov  bx,3
        mov  al,matr[bx][di]
        mov  cx,matr[bx][di]
        mov  ax,matr[bx*4][di]

;  ПРОВЕРКА РЕЖИМОВ АДРЕСАЦИИ С УЧЕТОМ СЕГМЕНТОВ
;  Переопределение сегмента
;  ------ вариант 1
        mov  ax, SEG vec2
        mov  es, ax
        mov  ax, es:[bx]
        mov  ax, 0
;  ------ вариант 2
        mov  es, ax
        push ds
        pop  es
        mov  cx, es:[bx-1]
        xchg cx,ax
;  ------ вариант 3
        mov  di,ind
        mov  es:[bx+di],ax
;  ------ вариант 4
        mov  bp,sp
        mov  ax,matr[bp+bx]
        mov  ax,matr[bp+di+si]
;  Использование сегмента стека
        push  mem1
        push  mem2
        mov   bp,sp
        mov   dx,[bp]+2
        ret   2
Main     ENDP
CODE     ENDS
        END Main
```

Отчет по работе должен содержать:

1) текст задания;
2) текст исходного файла программы с заданными значениями исходных данных;
3) описание обнаруженных при первоначальной трансляции ошибок и их объяснение;
4) листинг успешной трансляции программы с закомментированными ошибочными операторами;

5) протокол работы на компьютере, включающий описание выполнения каждой команды в пошаговом режиме под управлением отладчика, представленный в виде таблицы 1 (черновики протоколов должны быть подписаны преподавателем).
6) выводы по работе.

Лабораторная работа 3.

Представление и обработка целых чисел. Организация ветвящихся процессов

Задание

Разработать на языке Ассемблера программу, которая по заданным целочисленным значениям параметров a, b, i, k вычисляет:

a) значения функций i1 = f1(a,b,i) и i2 = f2(a,b,i);

b) значения результирующей функции res = f3(i1,i2,k),

где вид функций f1 и f2 определяется из табл. 2, а функции f3 - из табл.3 по цифрам шифра индивидуального задания (n1,n2,n3), приведенным в табл.4.

Значения a, b, i, k являются исходными данными, которые должны выбираться студентом самостоятельно и задаваться в процессе исполнения программы в режиме отладки. При этом следует рассмотреть всевозможные комбинации параметров a, b и k, позволяющие проверить различные маршруты выполнения программы, а также различные знаки параметров a и b.

Таблица 2	Таблица 3
f1 = / 15-2*i , при a>b \ 3*i+4 , при a<=b	f1 = / min(i1,i2), при k=0 \ max(i1,i2), при k/=0
f2 = / - (4*i+3) , при a>b \ 6*i -10 , при a<=b	f2 = / max(i1,10-i2), при k<0 \ |i1 - i2| , при k>=0
f3 = / 7 - 4*i , при a>b \ 8 -6*i , при a<=b	f3 = / |i1 + i2|, при k=0 \ min(i1,i2), при k/=0
f4 = / -(6*i - 4) , при a>b \ 3*(i+2) , при a<=b	f4 = / min (|i1 - i2|, 2), при k<0 \ max(-6, -i2), при k>=0
f5 = / 20 - 4*i , при a>b \ -(6*I - 6), при a<=b	f5 = / min(|i1|, 6), при k=0 \ |i1|+|i2|, при k/=0
f6 = / 2*(i+1) -4 , при a>b \ 5 - 3*(i+1), при a<=b	f6 = / |i1 - i2|, при k<0 \ max(7, |i2|), при k>=0
f7 = / -(4*i -5) , при a>b \ 10 - 3*i , при a<=b	f7 = / |i1| + |i2|, при k<0 \ max(6, |i1|), при k>=0
f8 = / - (6*i+8) , при a>b \ 9 -3*(i-1), при a<=b	f8 = / |i1| - |i2|, при k<0 \ max(4,|i2|-3), при k>=0

Таблица 4

№ студента	Шифр задания	№ студента	Шифр задания
1	1.2.1	14	3.4.2
2	1.3.2	15	3.5.3
3	1.4.3	16	3.6.4
4	1.5.4	17	3.7.5
5	1.6.5	18	3.8.6
6	1.7.6	19	4.5.7
7	1.8.7	20	4.6.8
8	2.3.8	21	4.7.2
9	2.4.7	22	4.8.3
10	2.5.6	23	5.6.4
11	2.6.5	24	5.7.5
12	2.7.4	25	5.8.6
13	2.8.3	26	6.8.1

Замечания:

1) при разработке программы нельзя использовать фрагменты, представленные на ЯВУ, в частности, для ввода-вывода данных. Исходные данные должны вводиться, а результаты контролироваться в режиме отладки;
2) при вычислении функций f1 и f2 вместо операции умножения следует использовать арифметический сдвиг и, возможно, сложение;
3) при вычислении функций f1 и f2 нельзя использовать процедуры;
4) при разработке программы следует минимизировать длину кода, для чего, если надо, следует преобразовать исходные выражения для вычисления функций.

Лабораторная работа 4.

Представление и обработка символьной информации с использованием строковых команд.

Задание

Разработать программу обработки символьной информации, реализующую функции:

- инициализация (вывод титульной таблички с указанием вида преобразования и автора программы) - на ЯВУ;
- ввода строки символов, длиной не более Nmax (<=80), с клавиатуры в заданную область памяти - на ЯВУ; если длина строки превышает Nmax, остальные символы следует игнорировать;
- выполнение заданного в таблице 5 преобразования исходной строки с записью результата в выходную строку - на Ассемблере;
- вывода результирующей строки символов на экран и ее запись в файл - на ЯВУ.

Ассемблерную часть программы включить в программу на ЯВУ по принципу встраивания (in-line).

Таблица 5.
Варианты заданий вида преобразования

1. Формирование выходной строки только из цифр и русских букв входной строки.
2. Формирование выходной строки только из цифр и латинских букв входной строки.
3. Формирование выходной строки только из русских и латинских букв входной строки.
4. Преобразование всех заглавных латинских букв входной строки в строчные, а восьмеричных цифр в инверсные, остальные символы входной строки передаются в выходную строку непосредственно.
5. Преобразование всех строчных латинских букв входной строки в заглавные, а десятичных цифр в инверсные, остальные символы входной строки передаются в выходную строку непосредственно.
6. Инвертирование введенных во входной строке цифр в десятичной системе счисления (СС) и преобразование строчных русских букв в заглавные, остальные символы входной строки передаются в выходную строку непосредственно.
7. Инвертирование введенных во входной строке цифр в восьмеричной СС и преобразование заглавных русских букв в строчные, остальные символы входной строки передаются в выходную строку непосредственно.
8. Преобразование введенных во входной строке шестнадцатиричных цифр в десятичную СС, остальные символы входной строки передаются в выходную строку непосредственно.
9. Преобразование введенных во входной строке десятичных цифр в восьмеричную СС, остальные символы входной строки передаются в выходную строку непосредственно.
10. Преобразование введенных во входной строке шестнадцатиричных цифр в двоичную СС, остальные символы входной строки передаются в выходную строку непосредственно.
11. Преобразование введенных во входной строке десятичных цифр в двоичную СС, остальные символы входной строки передаются в выходную строку непосредственно.
12. Формирование номера введенной латинской буквы по алфавиту и номера позиции его первого вхождения во входной строке и выдача их на экран.
13. Формирование номера введенной русской буквы по алфавиту и номера позиции его первого вхождения во входной строке и выдача их на экран.
14. Исключение латинских букв и цифр, введенных во входной строке при формировании выходной строки.
15. Исключение русских букв и цифр, введенных во входной строке, при формировании выходной строки.
16. Преобразование введенных во входной строке русских букв в латинские в соответствие с правилами транслитерации, остальные символы входной строки передаются в выходную строку непосредственно.
17. Преобразование введенных во входной строке латинских букв в русские в соответствие с правилами транслитерации, остальные символы входной строки передаются в выходную строку непосредственно.
18. Заменить введенные во входной строке русские буквы на десятичные числа, соответствующие их номеру по алфавиту, остальные символы входной строки передать в выходную строку непосредственно.
19. Заменить введенные во входной строке латинские буквы на десятичные числа, соответствующие их номеру по алфавиту, остальные символы входной строки передать в выходную строку непосредственно.
20. Заменить введенные во входной строке русские буквы на числа, соответствующие их номеру по алфавиту, представленному в шестнадцатиричной СС, остальные символы входной строки передать в выходную строку непосредственно.
21. Заменить введенные во входной строке латинские буквы на числа, соответствующие их номеру по алфавиту, представленному в шестнадцатиричной СС, остальные символы входной строки передать в выходную строку непосредственно.

22. Преобразование всех заглавных латинских букв входной строки в строчные, а десятичных цифр в инверсные, остальные символы входной строки передаются в выходную строку непосредственно.
23. Преобразование всех строчных латинских букв входной строки в заглавные, а шестнадцатиричных цифр в инверсные, остальные символы входной строки передаются в выходную строку непосредственно.
24. Инвертирование введенных во входной строке цифр в шестнадцатиричной системе счисления (СС) и преобразование строчных русских букв в заглавные, остальные символы входной строки передаются в выходную строку непосредственно.
25. Инвертирование введенных во входной строке цифр в десятичной СС и преобразование заглавных русских букв в строчные, остальные символы входной строки передаются в выходную строку непосредственно.

Замечания:
1) При выполнении преобразования обязательно использовать команды работы со строками;
2) При выполнении преобразования нельзя портить входную строку. Результат преобразования должен записываться в выходную строку.

Лабораторная работа 5.
Разработка собственного прерывания.

1. Краткие сведения.

Прерывание - это процесс вызова процедур для выполнения некоторой задачи, обычно связанной с обслуживанием некоторых устройств (обработка сигнала таймера, нажатия клавиши и т.д.). Когда возникает прерывание, процессор прекращает выполнение текущей программы (если ее приоритет ниже) и запоминает в стеке вместе с регистром флагов адрес возврата(CS:IP) - места, с которого будет продолжена прерванная программа. Затем в CS:IP загружается адрес программы обработки прерывания и ей передается управление. Адреса 256 программ обработки прерываний, так называемые векторы прерывания, имеют длину по 4 байта (в первых двух хранится значение IP , во вторых - CS) и хранятся в младших 1024 байтах памяти. Программа обработки прерывания должна заканчиваться инструкцией IRET (возврат из прерывания), по которой из стека восстанавливается адрес возврата и регистр флагов.

Программа обработки прерывания - это отдельная процедура, имеющая структуру:

```
SUBR_INT  PROC FAR
        PUSH AX   ; сохранение изменяемых регистров
         ...
        <действия по обработке прерывания>
        POP  AX   ; восстановление регистров
         ...
        MOV  AL, 20H
        OUT  20H,AL
        IRET
SUBR_INT  ENDP
```

Две последние строки обработчика прерывания, указанные перед командой IRET выхода из прерывания, необходимы для разрешения обработки прерываний с более низкими уровнями, чем только что обработанное.

Замечание: в лабораторной работе действиями по обработке прерывания может быть вывод на экран некоторого текста, вставка цикла задержки в вывод сообщения или включение звукового сигнала.

Программа, использующая новые программы обработки прерываний при своем завершении должна восстанавливать оригинальные векторы прерываний. Функция 35 прерывания 21H возвращает текущее значение вектора прерывания, помещая значение сегмента в ES, а смещение в BX. В соответствии с этим, программа должна содержать следующие инструкции:

```
; -- в сегменте данных
    KEEP_CS DW 0 ; для хранения сегмента
    KEEP_IP DW 0 ; и смещения вектора прерывания

; -- в начале программы
    MOV  AH, 35H   ; функция получения вектора
    MOV  AL, 1CH   ; номер вектора
    INT  21H
    MOV  KEEP_IP, BX  ; запоминание смещения
    MOV  KEEP_CS, ES  ; и сегмента вектора прерывания
```

Для установки адреса нового обработчика прерывания в поле векторов прерываний используется функция 25H прерывания 21H, которая помещает заданные адреса сегмента и смещения обработчика в вектор прерывания с заданным номером.

```
    PUSH DS
    MOV  DX, OFFSET ROUT ; смещение для процедуры в DX
    MOV  AX, SEG ROUT    ; сегмент процедуры
    MOV  DS, AX          ; помещаем в DS
    MOV  AH, 25H         ; функция установки вектора
    MOV  AL, 60H         ; номер вектора
    INT  21H             ; меняем прерывание
    POP  DS
```

Далее может выполняться вызов нового обработчика прерывания.

В конце программы восстанавливается старый вектор прерывания

```
    CLI
    PUSH DS
    MOV  DX, KEEP_IP
    MOV  AX, KEEP_CS
    MOV  DS, AX
    MOV  AH, 25H
    MOV  AL, 1CH
    INT  21H         ; восстанавливаем старый вектор прерывания
    POP  DS
    STI
```

Варианты заданий

Шифры, определяющие варианты заданий приведены в таблице 6.

Таблица 6

N бригады	Шифр задания	N бригады	Шифр задания
1	1A	7	3A
2	1B	8	3B
3	1C	9	3C
4	2A	10	4A
5	2B	11	4B
6	2C	12	4C

Цифра в шифре задает номер и назначение заменяемого вектора прерывания:

1 - 1Ch - прерывание от часов - генерируется автоматически операционной системой 18 раз в сек;

2 - 60h - прерывание пользователя - должно генерироваться в программе;

3 - 23h - прерывание, генерируемое при нажатии клавиш Control+C ;

4 - 08h - прерывание от системного таймера - генерируется автоматически операционной системой 18 раз в сек.

Буква определяет действия, реализуемые программой обработки прерываний:

A - Печать сообщения на экране;

B - Выдача звукового сигнала;

C - Приостановить вывод на экран (вставить цикл задержки).

Замечание: для исключения возможного взаимного влияния системных и пользовательских прерываний рекомендуется отвести в программе под стек не менее 1К байт.

Лабораторная работа 6.

Организация связи Ассемблера с ЯВУ на примере программы построения частотного распределение попаданий псевдослучайных целых чисел в заданные интервалы.

На языке высокого уровня (Pascal или C) генерируется массив псевдослучайных целых чисел, изменяющихся в заданном диапазоне и имеющих равномерное распределение. Необходимые датчики псевдослучайных чисел находятся в каталоге Tasks\RAND_GEN (при его отсутствии программу датчика получить у преподавателя).

Далее должен вызываться ассемблерный модуль(модули) для формирования распределения количества попаданий псевдослучайных целых чисел в заданные интервалы. В общем случае интервалы разбиения диапазона изменения псевдослучайных чисел могут иметь различную длину.

Результирующий массив частотного распределения чисел по интервалам, сформированный на ассемблерном уровне, возвращается в программу, реализованную на ЯВУ, и затем сохраняется в файле и выводится на экран средствами ЯВУ.

Исходные данные.

1. Длина массива псевдослучайных целыхчисел - NumRanDat (<= 16K, К=1024)
2. Диапазон изменения массива псевдослучайных целых чисел [Xmin, Xmax] , значения могут быть биполярные;

3. Количество интервалов, на которые разбивается диапазон изменения массива псевдослучайных целых чисел - NInt (<=24)
4. Массив левых границ интервалов разбиения LGrInt (должны принадлежать интервалу [Xmin, Xmax]).

Результаты:

1. Текстовый файл, строка которого содержит:
   - номер интервала,
   - левую границу интервала,
   - количество псевдослучайных чисел, попавших в интервал.

   Количество строк равно числу интервалов разбиения.
2. График, отражающий распределение чисел по интервалам.
   (необязательный результат)

В зависимости от номера бригады формирование частотного распределения должно производиться по одному из двух вариантов:

1. Для бригад с нечетным номером: подпрограмма формирования распределения количества попаданий псевдослучайных целых чисел в заданные интервалы реализуется в виде одного ассемблерного модуля, сразу формирующего требуемое распределение и возвращающего его в головную программу, написанную на ЯВУ;

2. Для бригад с четным номером: подпрограмма формирования распределения количества попаданий псевдослучайных целых чисел в заданные интервалы реализуется в виде двух ассемблерных модулей, первый из которых формирует распределение исходных чисел по интервалам единичной длины и возвращает его в вызывающую программу на ЯВУ как промежуточный результат. Это распределение должно выводиться в текстовом виде для контроля. Затем вызывается второй ассемблерный модуль, который по этому промежуточному распределению формирует окончательное распределение псевдослучайных целых чисел по интервалам произвольной длины (с заданными границами). Это распределение возвращается в головную программу и выдается как основной результат в виде текстового файла и, возможно, графика.

Лабораторная работа 7.

Использование арифметических операций над целыми числами и процедур в Ассемблере.

Разработать на языке Ассемблер процессора IntelX86 две процедуры:

- одна – выполняет прямое преобразование целого числа, заданного в регистре AX (или в паре регистров DX:AX) в строку, представляющую его символьное изображение в заданной системе счисления (с учетом или без учета знака в зависимости от варианта задания);
- другая - обратное преобразование строки, представляющей символьное изображение числа в заданной системе счисления в целое число, помещаемое в регистр AX (или в пару регистров DX:AX)

Строка должна храниться в памяти, а также выводиться на экран для индикации.

Отрицательные числа при представлении с учетом знака должны в памяти храниться в дополнительном коде, а на экране изображаться в прямом коде с явным указанием знака или в символьном виде со знаком.

Пример для однобайтовых чисел:

Десятичное число в символьном виде	Двоично-десятичное упакованное число
+ 35	00110101
- 35	11001011

Варианты заданий для выполнения преобразования определяются шифром, выбираемым по табл.7 и состоящим из 3-х цифр:

- 1-я цифра задает длину целого числа:
  16 бит, 2- 32 бита;
- 2-я цифра задает вид представления числа:
  1- с учетом знака, 2- без учета знака;
- 3-я цифра задает систему счисления для символьного изображения числа:
  1- двоичная, 2- восьмеричная, 3-десятичная, 4- шестнадцатиричная.

Таблица 7

№ бригады	Шифр Задания	№ бригады	Шифр Задания
1	2.1.1	7	1.1.4
2	2.2.1	8	2.2.4
3	1.2.3	9	2.1.3
4	1.1.3	10	2.2.3
5	2.2.2	11	2.1.4
6	1.1.2	12	1.1.1

2 Написать простейшую головную программу для иллюстрации корректности выполнения заданных преобразований. При этом вызываемые процедуры могут быть одного из следующих типов:

1 - near, 2 – far (в данном сегменте), 3 – far (в другом сегменте).

Связь по данным между основной программой и подпрограммами может осуществляться следующими способами:

A - только через РОНы;
B - через РОНы и общедоступные переменные;
C - через кадр стека.

Шифры, определяющие типы процедур и способы связи по данным, приведены в табл.8.

Таблица 8

№ бригады	Шифр Задания	№ бригады	Шифр Задания
1	1A2B	7	2C1A
2	1B3C	8	2A3B
3	1C2B	9	1A3B
4	2A1B	10	1C2A
5	2B2A	11	1A3C
6	2B3C	12	2C3B

Лабораторная работа N8

Обработка вещественных чисел. Программирование математического сопроцессора.

Разработать подпрограмму на языке Ассемблера, обеспечивающую вычисление заданной математической функции с использованием математического сопроцессора. Подпрограмма должна вызываться из головной программы, разработанной на языке С. При этом должны быть обеспечены заданный способ вызова и обмен параметрами. Альтернативный вариант реализации: разработать на языке Ассемблера фрагмент программы, обеспечивающий вычисление заданной математической функции с использованием математического сопроцессора, который включается по принципу in-line в программу, разработанную на языке С.

Возможный пример требуемой разработки программы для случая вычисления функции fmod(x,y)=x mod y приведен ниже в приложении 1.

Выполнить трансляцию программы с подготовкой ее ассемблерной версии и отладочной информации. Для выбранного контрольного набора исходных данных прогнать программу под управлением отладчика. При этом для каждой команды сопроцессора следует фиксировать содержимое используемых ячеек памяти, регистров ЦП и численных регистров сопроцессора до и после выполнения этой команды.
Проверить корректность выполнения вычислений для нескольких наборов исходных данных.

Примечание: При разработке программ вычисления заданной функции можно воспользоваться версиями соответствующих программ, взятыми из каталога FLOW_PNT.

В А Р И А Н Т Ы З А Д А Н И Й

ВАРИАНТ 1.

```
* function
Name          poly - generates a polynomial from arguments
Usage         double  poly(double x, int n, double c []);
Prototype in    math.h
Description    poly  generates  a  polynomial  in  x,  of  degree  n, with
           coefficients c[0],  c[1], ..., c[n].
For example, if  n=4  the generated polynomial is
                  c[4].x^4 + c[3].x^3 + c[2].x^2] + c[1].x + c[0]
The polynomial is calculated using Horner's method:
                  polynom = (..((x.c[n] + c[n-1]).x + c[n-2])..).x + c[0]
Return value    poly returns the  value of the polynomial as  evaluated for  the given x.
 *-------------------------------------------------------------------------*/
```

ВАРИАНТ 2.

```
* function
Name          cosh - hyperbolic function:
Usage         double cosh(double x);
Prototype in    math.h
Description     cosh computes the hyperbolic cosine of the input value.
                    cosh (x) = (exp(x) + exp(-x)) / 2
                    cosh is more accurately calculated by the polynomial  (1 + x^2/2)
                    when x is tiny (|x| < 2^-13).
```

ВАРИАНТ 3.

* function
Name sinh - hyperbolic sine function
Usage double sinh(double x);
Prototype in math.h
Description sinh computes the hyperbolic sine of the input value.
sinh (x) = (exp(x) - exp(-x)) / 2
but there is a loss of precision in using this formula directly near 0.
Since sinh(-x) = -sinh(x), compute sinh(|x|) and adjust the sign later.
If 0 <= x < 2^-33, return x.
If x >= .3465, use y = exp(x) and sinh(x) = (y - 1/y)/2
If 2^-33 <= x < .3465, use y = exp(x) – 1 and sinh(x) = (y + y/(1 + y))/2

ВАРИАНТ 4.

function
Name tanh - hyperbolic tangent function
Usage double tanh(double x);
Prototype in math.h
Description tanh calculates the hyperbolic tangent of the input.
tanh(x) = (exp(x) - exp(-x))/(exp(x) + exp(-x))/
For large arguments (magnitude greater than 32) the result will be +1.0 or -1.0.
Since tanh(-x) = -tanh(x), compute tanh(|x|) and adjust the sign later.
If 0 <= x < 2^-33, return x.
If x >= 32 return 1.
If x >= .17325, use y = exp(x) and tanh(x) = (y - 1/y)/(y + 1/y)
If 2^-33 <= x < .17325, use y = exp(2x) – 1 and sinh(x) = y/(2 + y)

ВАРИАНТ 5.

* function
Name ldexp - calculates value * 2^exp
Usage double ldexp(double value, int exp);
Prototype in math.h
Description ldexp calculates value * 2^exp

ВАРИАНТ 6.

* function
Name Acos - compute acos
Usage double Acos (double *xP);
Prototype in math.h
Description Computes acos of the number pointed to by xP.
Arguments to acos must be in the range -1 to 1, acos returns a value in the range 0 to pi.
Use the trig identities acos (x) = atan (sqrt (1-x^2) / x) */

ВАРИАНТ 7.

* function
Name Asin - compute asin
Usage double Asin (double *xP);
Prototype in math.h
Description Computes asin of the number pointed to by xP.
Arguments to asin must be in the range -1 to 1, asin returns a value in the range -pi/2 to pi/2.
Use the trig identities: asin (x) = atan (x / sqrt (1 - x^2)) ;

П Р И Л О Ж Е Н И Е 1.
Пример, иллюстрирующий выполнение задания

```
* function
 *       fmod - вычисляет x по модулю y, остаток от  x/y
 *-------------------------------------------------------------------------*/
#pragma inline
#include <asmrules.h>
#include <_math.h>
#include <math.h>
Имя            fmod - вычисляет x по модулю y,  остаток от  x/y
Использование   double fmod(double x, double y);
Прототип в...   math.h
Описание        fmod вычисляет  x - (y * int(x / y));
*-------------------------------------------------------------------------*/
#pragma warn -rvl
double  _FARFUNC fmod (double x, double y)
{
asm     FLD     DOUBLE (y)
asm     mov     ax, y [6]
asm     shl     ax, 1             /* Игнорируется знаковый бит */
asm     jz      mod_resultZero     /* Если делитель равен 0 */
asm     cmp     ax, 0FFE0h
asm     jnb     mod_isX           /* Если делитель - бесконечность */
asm     FLD     DOUBLE (x)
asm     mov     ax, x [6]
asm     shl     ax, 1
asm     jz      mod_xZero         /* Если делимое равно 0  */
asm     cmp     ax, 0FFE0h
asm     jnb     mod_overflow       /* Если делимое - бесконечность */
mod_keepTrying:
asm     FPREM
asm     push    bx
asm     mov     bx, sp
asm     FSTSW   W0 (SS_ [bx])      /* C2 будет установлен, если еще не конец */
asm     FWAIT
asm     pop     ax
asm     sahf
asm     jp      mod_keepTrying     /* C2 бит отображается на флаг четности */
asm     FSTP    st(1)             /* сбросить  делитель */
mod_end:
       return;
}
#pragma warn .rvl
```

Лабораторная работа N9

Программирование устанавливаемых драйверов символьного типа.

Состав устанавливаемого драйвера символьного типа и структура основных элементов.

Устанавливаемые драйверы бывают двух типов: блочные и символьные.

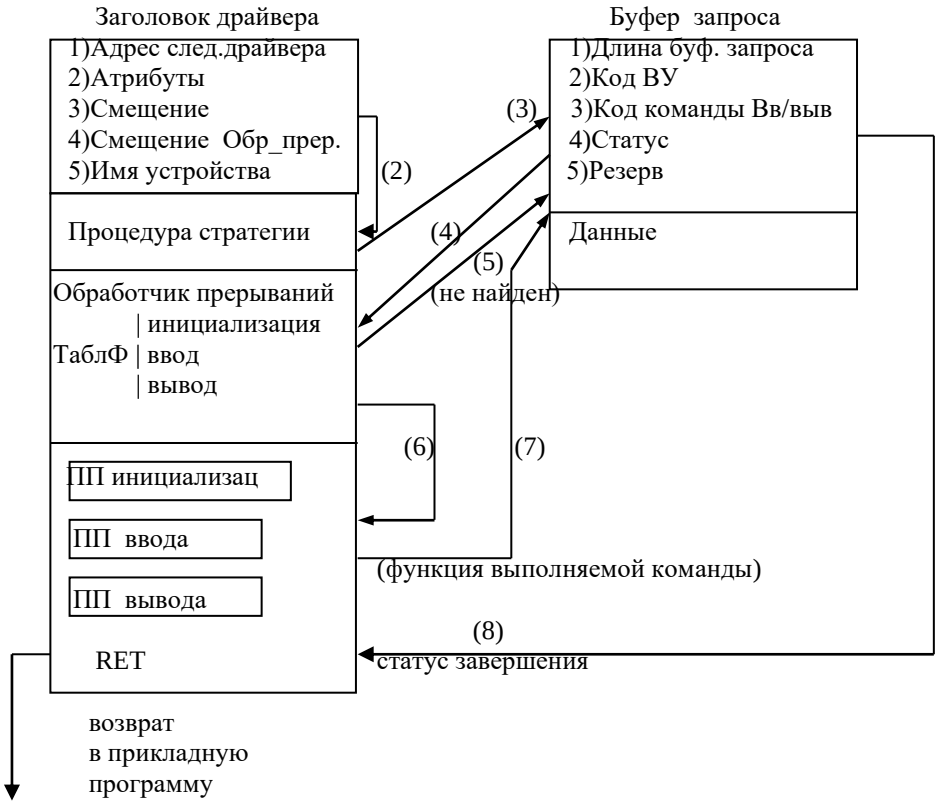
Первые используют файловую организацию и передачу данных блоками (обычно применяются для работы с дисковыми накопителями). Вторые используют посимвольную передачу данных (она проще) и применимы к любым внешним устройствам.

Символьный драйвер состоит из следующих элементов:

1. Заголовок драйвера.
2. Процедура стратегии.
3. Буфер запроса.
4. Обработчик прерываний подключает таблицу функций с набором операций, которые могут выполняться данным устройством.

Заголовок драйвера (18 байт) содержит:

1. Адрес следующего драйвера (4 байт).
2. Атрибуты (2 байт).
3. Смещение процедуры стратегии (2 байт).
4. Смещение обработчика прерываний.
5. Имя устройства (8 байт).



Драйвер обычно записывается как самостоятельный модуль, но без PSP, поэтому не может запускаться самостоятельно.

1. Поле «Атрибуты» содержит:

15 бит : 1- символьный драйвер, 0 – блочный драйвер
14 бит : поддержка IOCTL
13 бит : формат блоков
1 – IBM
0 – любой другой
……
3 бит : 1 – часы , 0 – не часы
2 бит : 1 – null, 0 – не null
1 бит : 1 – STDOUT (стандарт вывода)
0 - STDIN (стандарт ввода)

2. Процедура стратегии выполняются только один раз на этапе загрузки и служит для запоминания длинного указателя на буфер запроса, создаваемый для драйвера самой операционной системой.

```
DEV_STRAT:
mov cs:SEG_PQBF, es  ;
mov cs:OFF_PQBF, es  ;
ret
SEG_PQBF  DW ?
OFF_PQBF  DW ?
```

3. Буфер запроса.

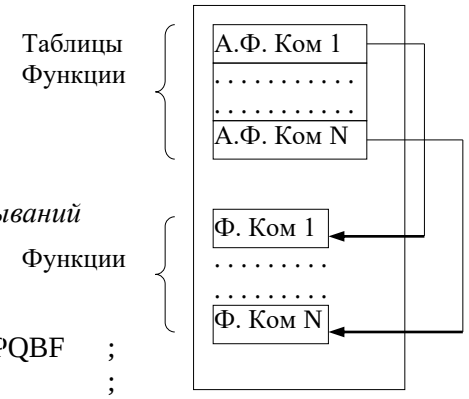
Структура данных, через которые прикладная программа связывается с драйвером (прикладная программа задает вид операции ввода-вывода и место расположения данных, а драйвер возвращает ей свой статус (все идет через буфер запроса)).

Буфер состоит их стандартной части 13 байт, называемой заголовок, и последующей части, называемой данные, содержащей различную информацию в зависимости от вида драйвера и исполняемой операции ввода-вывода.

Заголовок запроса.
+0 : длина буфера запроса
+1 : код внешнего устройства
+2 : код команды ввода-вывода
+3 : статус
+5 : резерв
+13d : данные

4. Обработчик прерывания.

Это только процедура, которая таблично вызывает функцию, реализующую команду ввода-вывода; она завершается командой RET и выдачей статуса.



Код обработчика прерываний

```
dev_intpr:
        pushA
        mov ax, SEG_PQBF     ;
        mov es, ax           ;
        mov bx, OFF_PQBF     ;
        mov al, es:[bx+2]    ;
        shl al, 1            ;
        xor ah, ah           ; Подготовка и загрузка
        lea di, FUN_TAB      ;  смещения функции
        add di, ax           ; в таблицу функции
        jmp word ptr[di]     ;
        FUN_TAB LABEL WORD;
                DW INITIALIZE
                DW CHECK_MEDIA
                . . . . . .
                DW INPUT_DATA
Тринадцать      . . . . . .
функций         DW OUPUT_DATA
```

```
        . . . . . .
        DW OUTPUT_STATUS
        . . . . . .
        DW IOCTL_OUT
```

Определенная функция находится в определенной строке таблицы.
Предположим что драйвер поддерживает только две функции init и out, тогда получим:

```
CHECK_MEDIA:
. . . . . .
INPUT_DATA:
. . . . . .
IOCTL_OUT:
or es.word ptr[bx]+3, 8103h
JMP QUIT
INITIALIZE:
lea ax, E_O_P
mov es, word ptr[bx]+14, ax
mov es, word ptr[bx]+16, cs
JMP QUIT
OUTPUT_DATA
QUIT:
OR es:word ptr[bx]+3, 100h
POPA
RET
```

В поле статус, бит
15 – ошибка
9 – драйвер занят
8 – функция завершена
с 0 – 7 – код ошибки

Пример: 8103h означает 8 – ошибка , 1 – операция завершена, 03 – неизвестная ошибка.
После разработки драйвера нужно вставить его в config.sys.

Задание на выполнение работы.

1. Разработать программу устанавливаемого драйвера символьного типа, реализующую функции:
- инициализация (фиксация границы драйвера и вывод титульной таблички);
- ввод с клавиатуры строки символов, длиной не более Nmax;
- выполнение заданного в таблице 9 преобразования введенной строки и вывода результирующей строки символов на экран.

2. Разработать на языке С тестирующую программу для проверки функционирования драйвера.

Таблица 9
Варианты заданий вида преобразования.

1. Формирование выходной строки только из цифр введенной строки.
2. Формирование выходной строки только из латинских букв введенной строки.
3. Преобразование всех заглавных латинских букв входной строки в строчные.
4. Преобразование всех строчных латинских букв входной строки в заглавные.
5. Инвертирование введенных во входной строке цифр в десятичной системе счисления.

6. Формирование номера введенной латинской буквы по алфавиту и выдача на экран последовательности номеров, разделенных пробелами.

7. Игнорирование (исключение) строчных латинских букв, введенных во входной строке при формировании выходной строки.

8. Формирование выходной строки только из русских букв введенной строки.