Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет им. Г.И. Носова»
Многопрофильный колледж

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

ПМ. 01. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ КОМПЬЮТЕРНЫХ СИСТЕМ

по специальности 09.02.03 Программирование в компьютерных системах
базовой подготовки

Магнитогорск, 2017

ОДОБРЕНО:

Предметно - цикловой комиссией *«Информатики и вычислительной техники»*
Председатель И.Г. Зорина
Протокол № 07 от 14 марта 2017г.

Методической комиссией МпК
Протокол №4 от «23» марта 2017г

Разработчик (и):
преподаватели МпК
ФГБОУ ВО «МГТУ» Людмила Александровна Фетисова
Власта Диляуровна Тутарова

Методические указания по выполнению практических занятий разработаны на основе рабочей программы ПМ. 01. Разработка программных модулей программного обеспечения для компьютерных систем.

Содержание практических работ ориентировано на формирование общих и профессиональных компетенций по основной профессиональной образовательной программе по специальности 09.02.03 Программирование в компьютерных системах базовой подготовки:
МДК.01.01. Системное программирование ;
МДК.01.02. Прикладное программирование.

СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ

Важную часть теоретической и профессиональной практической подготовки студентов составляют практические занятия.

Состав и содержание практических работ направлены на реализацию действующего федерального государственного образовательного стандарта среднего профессионального образования по специальности 09.02.03 Программирование в компьютерных системах базовой подготовки.

Ведущей дидактической целью практических занятий является формирование практических умений - профессиональных (умений выполнять определенные действия, операции, необходимые в последующем в профессиональной деятельности) и/или учебных (умений решать задачи), необходимых в последующей учебной деятельности по профессиональным модулям.

В соответствии с рабочей программой ПМ. 01. Разработка программных модулей программного обеспечения для компьютерных систем МДК.01.01. Системное программирование, МДК.01.02. Прикладное программирование предусмотрено проведение практических работ.

В результате их выполнения, обучающийся должен:

уметь:

- осуществлять разработку кода программного модуля на современных языках программирования;
- создавать программу по разработанному алгоритму как отдельный модуль;
- выполнять отладку и тестирование программы на уровне модуля;
- оформлять документацию на программные средства;
- использовать инструментальные средства для автоматизации оформления документации;

Содержание практических занятий ориентировано на формирование общих компетенций по профессиональному модулю основной профессиональной образовательной программы по специальности:

- ОК 1. Понимать сущность и социальную значимость своей будущей профессии, проявлять к ней устойчивый интерес.
- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.

- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.
- ОК 6. Работать в коллективе и в команде, эффективно общаться с коллегами, руководством, потребителями.
- ОК 7. Брать на себя ответственность за работу членов команды (подчиненных), за результат выполнения заданий.
- ОК 8. Самостоятельно определять задачи профессионального и личностного развития, заниматься самообразованием, осознанно планировать повышение квалификации.
- ОК 9. Ориентироваться в условиях частой смены технологий в профессиональной деятельности.

И овладению профессиональными компетенциями:

ПК.1.1. Выполнять разработку спецификаций отдельных компонент.

ПК.1.2. Осуществлять разработку кода программного продукта на основе готовых спецификаций на уровне модуля.

ПК.1.3. Выполнять отладку программных модулей с использованием специализированных программных средств.

ПК.1.4. Выполнять тестирование программных модулей.

ПК.1.5. Осуществлять оптимизацию программного кода модуля.

ПК.1.6. Разрабатывать компоненты проектной и технической документации с использованием графических языков спецификаций.

ПК.1.7. Осуществлять работу с системой контроля версий.

Выполнение студентами практических работ по ПМ. 01. Разработка программных модулей программного обеспечения для компьютерных систем МДК.01.01. Системное программирование, МДК.01.02. Прикладное программирование направлено на:

- обобщение, систематизацию, углубление, закрепление, развитие и детализацию полученных теоретических знаний по конкретным темам междисциплинарных курсов;
- формирование умений применять полученные знания на практике, реализацию единства интеллектуальной и практической деятельности;
- формирование и развитие умений: наблюдать, сравнивать, сопоставлять, анализировать, делать выводы и обобщения;
- развитие интеллектуальных умений у будущих специалистов;

- выработку при решении поставленных задач профессионально значимых качеств, таких как самостоятельность, ответственность, точность, творческая инициатива.

Продолжительность выполнения практической работы составляет не менее двух академических часов и проводится после соответствующего занятия, которое обеспечивает наличие знаний, необходимых для ее выполнения.

2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ

МДК.01Системное программирование

Тема 01.01. Машинное представление данных

Практическое занятие № 1. Перевод чисел в различные системы счисления

Цель работы:

приобретение навыков выполнения операций в различных системах счисления.

Основная информация

Система счисления, или просто счисление, или нумерация, – набор конкретных знаков–цифр вместе с системой приемов записи, которая представляет числа этими цифрами.

1. Основные понятия систем счисления

Система счисления – это совокупность правил и приемов записи чисел с помощью набора цифровых знаков. Количество цифр, необходимых для записи числа в системе, называют основанием системы счисления. Основание системы записывается в справа числа в нижнем индексе: 5_{10}; 1110110_2; $AF178_{16}$.

Различают два типа систем счисления:

– позиционные, когда значение каждой цифры числа определяется ее позицией в записи числа;

– непозиционные, когда значение цифры в числе не зависит от ее места в записи числа.

Примером непозиционной системы счисления является римская: числа IX, IV, XV и т.д. Примером позиционной системы счисления является десятичная система, используемая повседневно.

Любое целое число в позиционной системе можно записать в форме многочлена:

$$X_S = \{A_n A_{n-1} \ldots A_2 A_1\} = A_n \cdot S^{n-1} + A_{n-1} \cdot S^{n-2} + \ldots + A_2 \cdot S^1 + A_1 \cdot S^0,$$

где S — основание системы счисления;

A_n — цифры числа, записанного в данной системе счисления;

n — количество разрядов числа.

Пример. Число 6293_{10} запишется в форме многочлена следующим образом:

$$6293_{10} = 6 \cdot 10^3 + 2 \cdot 10^2 + 9 \cdot 10^1 + 3 \cdot 10^0$$

Десятичная система счисления – в настоящее время наиболее известная и используемая. неправильное название удерживается и поныне.

Десятичная система использует десять цифр —– 0, 1, 2, 3, 4, 5, 6, 7, 8 и 9, а также символы “+” и “–” для обозначения знака числа и запятую или точку для разделения целой и дробной частей числа.

В вычислительных машинах используется двоичная система счисления, её основание — число 2. Для записи чисел в этой системе используют только две цифры — 0 и 1.

Таблица 1

Соответствие чисел, записанных в различных системах счисления

Десятичная	Двоичная	Восьмеричная	Шестнадцатеричная
1	001	1	1
2	010	2	2
3	011	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

2. Правила перевода чисел из одной системы счисления в другую

Перевод чисел из одной системы счисления в другую составляет важную часть машинной арифметики. Рассмотрим основные правила перевода.

Для перевода двоичного числа в десятичное необходимо его записать в виде многочлена, состоящего из произведений цифр числа и соответствующей степени числа 2, и вычислить по правилам десятичной арифметики:

$$X_2 = A_n \cdot 2^{n-1} + A_{n-1} \cdot 2^{n-2} + A_{n-2} \cdot 2^{n-3} + \ldots + A_2 \cdot 2^1 + A_1 \cdot 2^0$$

При переводе удобно пользоваться таблицей степеней двойки:

Таблица 2

Степени числа 2

n	0	1	2	3	4	5	6	7	8	9	10
2^n	1	2	4	8	16	32	64	128	256	512	1024

Пример. Число 11101000_2 перевести в десятичную систему счисления.

$$11101000_2 = 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 232_{10}$$

Для перевода восьмеричного числа в десятичное необходимо его записать в виде многочлена, состоящего из произведений цифр числа и соответствующей степени числа 8, и вычислить по правилам десятичной арифметики:

$$X_8 = A_n \cdot 8^{n-1} + A_{n-1} \cdot 8^{n-2} + A_{n-2} \cdot 8^{n-3} + \ldots + A_2 \cdot 8^1 + A_1 \cdot 8^0$$

При переводе удобно пользоваться таблицей степеней восьмерки:

Таблица 3

Степени числа 8

n	*0*	*1*	*2*	*3*	*4*	*5*	*6*
8^n	*1*	*8*	*64*	*512*	*4096*	*32768*	*262144*

Пример. Число 75013_8 перевести в десятичную систему счисления.

$$75013_8 = 7 \cdot 8^4 + 5 \cdot 8^3 + 0 \cdot 8^2 + 1 \cdot 8^1 + 3 \cdot 8^0 = 31243_{10}$$

1. Для перевода шестнадцатеричного числа в десятичное необходимо его записать в виде многочлена, состоящего из произведений цифр числа и соответствующей степени числа 16, и вычислить по правилам десятичной арифметики:

$$X_{16} = A_n \cdot 16^{n-1} + A_{n-1} \cdot 16^{n-2} + A_{n-2} \cdot 16^{n-3} + ... + A_2 \cdot 16^1 + A_1 \cdot 16^0$$

При переводе удобно пользоваться таблицей степеней числа 16:

Таблица 4

Степени числа 16

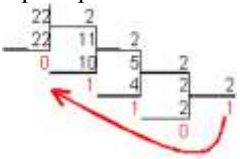
n	*0*	*1*	*2*	*3*	*4*	*5*	*6*
16^n	*1*	*16*	*256*	*4096*	*65536*	*1048576*	*16777216*

Пример. Число $FDA1_{16}$ перевести в десятичную систему счисления.

$$FDA1_{16} = 15 \cdot 163 + 13 \cdot 16^2 + 10 \cdot 16^1 + 1 \cdot 16^0 = 64929_{10}$$

Для перевода десятичного числа в двоичную систему его необходимо последовательно делить на 2 до тех пор, пока не останется остаток, меньший или равный 1. Число в двоичной системе записывается как последовательность последнего результата деления и остатков от деления в обратном порядке.

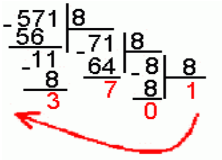
Пример. Число 22_{10} перевести в двоичную систему счисления.



$$22_{10} = 10110_2$$

Для перевода десятичного числа в восьмеричную систему его необходимо последовательно делить на 8 до тех пор, пока не останется остаток, меньший или равный 7. Число в восьмеричной системе записывается как последовательность цифр последнего результата деления и остатков от деления в обратном порядке.

Пример. Число 571_{10} перевести в восьмеричную систему счисления.

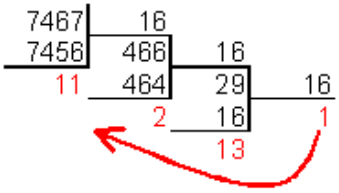


$$571_{10} = 1073_8$$

Для перевода десятичного числа в шестнадцатеричную систему его необходимо последовательно делить на 16 до тех пор, пока не останется остаток, меньший или равный 15. Число в

шестнадцатеричной системе записывается как последовательность цифр последнего результата деления и остатков от деления в обратном порядке.

Пример. Число 7467_2 перевести в шестнадцатеричную систему счисления.



$$7467_{10} = 1D2B_{16}$$

Чтобы перевести число из двоичной системы в восьмеричную, его нужно разбить на триады (тройки цифр), начиная с младшего разряда, в случае необходимости дополнив старшую триаду нулями, и каждую триаду заменить соответствующей восьмеричной цифрой (табл. 3).

Пример. Число 1001011_2 перевести в восьмеричную систему счисления.

$$001\ \ 001\ \ 011_2 = 113_8$$

Чтобы перевести число из двоичной системы в шестнадцатеричную, его нужно разбить на тетрады (четверки цифр), начиная с младшего разряда, в случае необходимости дополнив старшую тетраду нулями, и каждую тетраду заменить соответствующей восьмеричной цифрой (табл. 3).

Пример. Число 1011100011_2 перевести в шестнадцатеричную систему счисления.

$$0010\ \ 1110\ \ 0011_2 = 2E3_{16}$$

Для перевода восьмеричного числа в двоичное необходимо каждую цифру заменить эквивалентной ей двоичной триадой.

Пример. Число 531_8 перевести в двоичную систему счисления.

$$531_8 = 101011001_2$$

Для перевода шестнадцатеричного числа в двоичное необходимо каждую цифру заменить эквивалентной ей двоичной тетрадой.

Пример. Число $EE8_{16}$ перевести в двоичную систему счисления.

$$EE8_{16} = 111011101000_2$$

При переходе из восьмеричной системы счисления в шестнадцатеричную и обратно, необходим промежуточный перевод чисел в двоичную систему.

Пример 1. Число FEA_{16} перевести в восьмеричную систему счисления.

$$FEA_{16} = 111111101010_2$$

$$111\ 111\ 101\ 010_2 = 7752_8$$

Пример 2. Число 6653_8 перевести в шестнадцатеричную систему счисления.

$$6653_8 = 110110101011_2$$

$$1101\ 1010\ 1011_2 = DAB_{16}$$

Арифметические действия над целыми числами в 2-ой системе счисления :

1.Операция сложения выполняется с использованием таблицы двоичного сложения в одном разряде:

+	0	1
0	0	1
1	1	10

Пример.

а) +10012 б) +11012 в) +111112

10102 10112 12

100112 110002 1000002

2.Операция вычитания выполняется с использованием таблицы вычитания, в которой 1 обозначается заем в старшем разряде.

Пример.

а) –1011100112 б) –1101011012

1000110112 1010111112

0010110002 0010011102

3.Операция умножения выполняется по обычной схеме, применяемой в десятичной с/с с последовательным умножением множимого на очередную цифру множителя.

Пример.

а) × 110012 б) × 1012

11012 112

11001	101	×
11001	101	0
11001	11112	
1010001012		1

4.Операция деления выполняется по алгоритму, подобному алгоритму выполнения операции деления в 10–ой с/с.

Пример

1010001012	11012	1000110002	11112	
1101	110012	1111	100102	
1110	10100			
1101	1111			
001101	10102 –остаток			
1101				
0				

2.1. Сложение и вычитание в восьмеричной системе счисления.

При выполнении сложения и вычитания в 8-ой с/с необходимо соблюдать следующие правила:

в записи результатов сложения и вычитания могут быть использованы только цифры восьмеричного алфавита;

десяток восьмеричной системы счисления равен 8, т.е. переполнение разряда наступает, когда результат сложения больше или равен 8.

В этом случае для записи результата надо вычесть 8, записать остаток, а к старшему разряду прибавить единицу переполнения;

3)если при вычитании приходится занимать единицу в старшем разряде, эта единица переносится в младший разряд в виде восьми единиц.

Пример

+ 7708	+ 7508
2368	2368
12268	5128

2.2. Сложение и вычитание в шестнадцатеричной системе счисления.

При выполнении этих действий в 16–ой с/с необходимо соблюдать следующие правила:

1)при записи результатов сложения и вычитания надо использовать цифры шестнадцатеричного алфавита: цифры,

обозначающие числа от 10 до 15 записываются латинскими буквами, поэтому, если результат является числом из этого промежутка, его надо записывать соответствующей латинской буквой;

2)десяток шестнадцатеричной системы счисления равен 16, т.е. переполнение разряда поступает, если результат сложения больше или равен 16, и в этом случае для записи результата надо вычесть 16, записать остаток, а к старшему разряду прибавить единицу переполнения;

3)если приходится занимать единицу в старшем разряде, эта единица переносится в младший разряд в виде шестнадцати единиц.

Примеры.

+ B0916	+ B0916
TFA16	7FA16
1A0316	30F16

Задание

1. Выполнить перевод чисел

а) из 10–ой с/с в 2–ую систему счисления: 165; 541; 600; 720; 43,15; 234,99.

б) из 2–ой в 10–ую систему счисления: 110101_2; 11011101_2; 110001011_2; $1001001,111_2$

в) из 2–ой с/с в 8–ую ,16–ую с/с:
100101110_2; 100000111_2; 111001011_2; 1011001011_2; 110011001011_2; $10101,10101_2$; $111,011_2$

г) из 10–ой с/с в 8–ую, 16–ую с/с: 69; 73; 113; 203; 351; 641; 478,99; 555,555

д) из 8–ой с/с в 10–ую с/с: 35_8 ; 65_8 ; 215_8 ; 327_8 ; 532_8 ; 751_8; $45,454_8$

е) из 16–ой с/с в 10–ую с/с: $D8_{16}$; $1AE_{16}$; $E57_{16}$; $8E5_{16}$; FAD_{16}; $AFF,6A7_{16}$

2. Выпишите целые десятичные числа, принадлежащие следующим числовым промежуткам:

$[10101_2; 110000_2]$; $[14_8; 20_8]$; $[18_{16}; 30_{16}]$

3. Выполнить операции:

а) сложение в двоичной системе счисления

$+\ 10010011_2$	$+\ 1011101_2$	$+\ 10110011_2$	$+10111001{,}1_2$
1011011_2	11101101_2	1010101_2	$10001101{,}1_2$

б) вычитание в 2–ой системе счисления

$-\ 100001000_2$	$-\ 110101110_2$	$-\ 11101110_2$	$-10111001{,}1_2$
10110011_2	10111111_2	1011011_2	$10001101{,}1_2$

в) умножение в 2–ой системе счисления

$\times\ 100001_2$	$\times\ 100101_2$	$\times\ 111101_2$	$\times\ 11001{,}01_2$
111111_2	111011_2	111101_2	$11{,}01_2$

г) деление в 2–ой системе счисления

1) $111010001001_2 / 111101_2$
2) $100011011100_2 / 110110_2$
3) $10000001111_2 / 111111_2$

д) сложение 8–ых чисел

$+\ 715_8$	$+\ 524_8$	$+\ 712_8$	$+\ 321_8$	$+\ 5731_8$	$+\ 6351_8$
73_8	57_8	763_8	765_8	1376_8	737_8

е) вычитание 8–ых чисел

$-\ 137_8$	$-\ 436_8$	$-\ 705_8$	$-\ 538_8$	$-\ 7213_8$
72_8	137_8	76_8	57_8	537_8

ж) сложение 16–ых чисел

$+\ A13_{16}$	$+\ F0B_{16}$	$+\ 2EA_{16}$	$+\ ABC_{16}$	$+\ A2B_{16}$
$16F_{16}$	$1DA_{16}$	FCE_{16}	$C7C_{16}$	$7F2_{16}$

з) вычитание 16–ых чисел

$-\ À17_{16}$	$-\ DFA_{16}$	$-\ FO5_{16}$	$-\ DE5_{16}$	$-\ D3C1_{16}$
$1FC_{16}$	$1AE_{16}$	AD_{16}	AF_{16}	$D1F_{16}$

4. Вычислите выражение:

$(1111101_2 + AF_{16}) / 36_8$; $125_8 + 11101_2 \times A2_{16} / 1417_8$

Тема 01.01.03 Организация различных видов системной памяти
Практическое занятие Работа и использование отладчика AFDP**.**
№ 2 Описание интерфейса отладчика. Функциональные клавиши.

Цель работы:

Познакомиться с интерфейсом отладчика AFDP и его основными возможностями

Основная информация

Отладчик – это специальная программа, предназначенная для ввода и пошагового выполнения исполняемых программ. Существует ряд причин для использования данных программ разными категориями пользователей: школьниками, студентами и профессиональными программистами:

- ♦ С помощью отладчика можно по ходу выполнения программы просматривать исполняемый код, наблюдать за изменениями в регистрах процессора, отслеживать значения и изменения отдельных байтов и целых областей памяти, а также портов ввода/вывода и многое другое.
- ♦ Отладчик с учебной точки зрения – это довольно удобная модель процессора, позволяющая понять роль и работу отдельных его элементов «изнутри», а также функционирование и устройство всей компьютерной системы.
- ♦ Отладчик является замечательным инструментом, дающим возможность изучать исполняемый код своих и чужих программ. Самый лучший способ изучить отдельные команды — написать небольшую тестовую программу в мнемокодах, загрузить ее в отладчик и наблюдать за результатами ее работы в пошаговом режиме. Удивляет то количество информации, которое можно получить, просто наблюдая и анализируя работу машинных команд.

Всем, кто хочет понимать свои действия, любит управлять событиями и глубоко вникать в детали, для того отладчик – незаменимый инструмент.

Для реального режима работы процессора наиболее широко используются такие отладчики, как **DEBUG, AFD** и **Turbo Debug**. Эти средства по своим функциональным возможностям во многом схожи, а основные отличия заключаются в некоторых сервисных возможностях и организации интерфейса с пользователем.

Простейший отладчик DEBUG.exe входит в операционную систему MS DOS и Windows 9∗ в качестве внешней команды. У него полностью отсутствует интерфейсная оболочка, и работа осуществляется посредством командной строки в

3

текстовом режиме. После его запуска появляется характерное приглашение:

– (дефис) и если набрать вопросительный знак (?), то появляется помощь на русском языке в виде списка синтаксиса основных команд. В данных указаниях мы его рассматривать не будем из-за относительной

простоты этого отладчика и наличия подробного описания по его использованию во многих книгах, например [1-3].

Отладчик **Turbo Debug** – поставляется с большинством версий Borland C++ и Pascal. Подобно всем другим отладчикам, он работает в режиме супервизора, беря на себя управление программой, и позволяет по шагам исполнять код программы. Можно потребовать, чтобы Turbo Debug исполнял программу до некоторой точки или до появления определенной ситуации. Можно изменять значения в памяти, временно вставлять новые команды, а также менять значения регистров и флагов. Можно использовать Turbo Debug для ввода или редактирования программ в машинных кодах, если количество команд не слишком велико. Из-за наличия хорошего описания по его применению во многих книгах, например, [4-6], мы на его работе также останавливаться не будем.

Простым и довольно удобным отладчиком, имеющим интерфейсную оболочку, является AFD. Известны две его версии:

• **AFD** (Advanced Full screen Debug);

• **AFDP** (Advanced Full screen Debug Professional). Работа с ними во многом схожа, и мы не будем их различать, ориентируясь на более позднюю и удобную версию – AFDP.

1.1. Описание основных полей интерфейса

Программа AFD, предназначена для отладки выполняемых программ из файлов типа EXE или COM. Однако можно загрузить в память, просмотреть и при необходимости отредактировать любой другой файл.

Основное окно программы AFDP состоит из следующих **полей информации** (рис. 1):

- *состояния процессора,* включающее в себя содержимое пользовательских регистров, указатель команд, содержимое четырех слов стека (Staks),

регистра флагов (Flags) и значение восьми отдель-$_4$ ных флагов **(1)**;

- *командная строка* для ввода команд отладчика **(2);**
- размещение области памяти в виде текста программы в мнемониках Ассемблера **(3);**
- *поле* отображения содержимого дампа памяти *(Memory 1)* по восемь байт в строке с указанием полного адреса первого байта строки **(4);**
- *поле* отображения содержимого дампа памяти

Рис.1. Основное окно отладчика AFDP

(Memory 2) по 16 байт в строке в шестнадцатеричном, а также в символьном виде с указанием полного адреса первого байта строки **(5);**

- *поле подсказки* по использованию функциональных клавиш F1-F10 **(6).**

Внутри любого поля (1) – (5) можно изменять его содержимое: регистров, памяти, стека и состояние флагов. Для изменения соответствующего поля в него необходимо перевести курсор, который изначально находится в поле ввода команд. Перевод курсора осуществляется функциональными клавишами F7-↑, F8 -↓, F9 - , F10 – →.

Внутри активного поля курсор перемещается с помощью клавиш управления курсором и клавишей табуляции – Tab. Можно изменять любые данные в полях Memory 1 и Memory 2.

Необходимо иметь в виду, что отладчик оперирует только с шестнадцатеричными числами, которые являются компактным представлением двоичных чисел. Допустимыми значениями являются числа от 0 до F (можно писать как строчными, так и заглавными символами). Исключением является правая часть поля 5 символьного представления данных, где допустимы любые ASCII символы. Кроме того, отдельные флаги могут принимать лишь значения 0 или 1. При написании и редактировании программы все адреса, данные и константы записываются только в шестнадцатеричном виде.

1.2. Функциональные клавиши

В поле (6) подсказки приведены данные по использованию функциональных клавиш.

F1 (Step) – *пошаговое* выполнение одной машинной команды. При нажатии на эту клавишу выполняется текущая команда программы, указатель команд переходит на следующую команду и, соответственно, меняется содержимое регистров IP и CS. Вызов подпрограмм, работа циклов и выполнение прерываний обрабатывается по одной команде.

F2 (ProcStep) – *процедурное* выполнение программы. При нажатии на F2 выполняется текущая команда, как и в случае нажатия F1. Однако, если текущей командой является вызов процедуры (CALL), прерывания (INT) или цикла (LOOP), то за одно нажатие клавиши выполняется целиком процедура, прерывание или весь цикл. Это удобно для достижения различных целей отладки.

F3 (Retrieve) – просмотреть предыдущие выполненные команды. Это довольно удобно для повтора предыдущей команды. **F4 – (Help ON {OFF})** вызвать или убрать контекстную подсказку для текущей команды или просмотреть все доступные команды отладчика листая справочную информацию клавишами

Page Up и Page Dn. При нажатии F4 второе окно памяти (*Memory 2*) заменяется на окно подсказки для текущей команды. **F5 (BRK Menu) –** установка точек прерывания выполнения программы и переход к экрану установки точек прерывания. При этом функциональные клавиши F1, F3, F4, F7, F9, F10 начинают нести другой смысл. Выход из этого режима повторное нажатие **F5**

(CMD line).

F7 (up) – перевод курсора на вышестоящее поле;

F8 (dn) – перевод курсора на поле ниже;

F9 (le) – перевод курсора в левое поле от активного;

F10 (ri) – перевод курсора в правое поле от активного.

Тема 01.04. Основные машинные команды Ассемблер

Практическое занятие № 3. Работа и использование отладчика AFDP: Основные команды отладчика.

Цель работы:

Изучить основные команды отладчика

Основные команды отладчика

Команды набираются в поле ввода команд (2) и выполняются нажатием на клавишу Enter. Рассмотрим более подробно работу основных команд. В фигурных скобках указаны не обязательные операнды.

L {/p} {/addr} filename {parameter}

Команда используется для загрузки программы или данных из файла в память.

filename – полное имя файла включая путь к файлу по спецификации DOS; addr – адрес загрузки файла в память является необязательным параметром и представляет полный адрес либо только смещение относительно сегмента CS. При загрузке выполняемой программы формируется PSP со всеми необходимыми параметрами. В отдельных случаях можно указать адрес памяти, начиная с которого

будет загружаться и далее выполняться программа. По умолчанию для *.com-программы этот адрес определяется содержимым регистра CS и смещением 100h. Если файл благополучно загружен, то в регистровой паре (BX; CX) находится количество записанных в память байт программы.

/p – параметр для распаковки выполняемых файлов EX-
EPACK.

W { addr, length, filename }

Команда записывает содержимое области памяти в файл. По умолчанию сегментным регистром адреса принимается DS. Длина записывается как количество байт в формате четырехразрядного шестнадцатиричного числа.

{R} Регистр = Значение. Команда служит для установки содержимого регистра или флага, например, AX=11FF; AX=BX; FL=3202. Регистр флагов (FL) определяется как шестнадцатиразрядный. Для установки отдельных флагов используются следующие их названия: OF, DF, IF, SF, ZF, AF, PF и CF, например, CF=1,

ZF=1.

D addr. Команда служит для установки адреса области памяти, которая отображается в поле размещения текста программы.

Сегментный регистр по умолчанию CS.

Mn Адрес/Регистр. Команда определяет адрес памяти, отображаемой в одном из двух полей содержимого памяти (n=1 или n=2). Сегментный регистр указывается в соответствующем поле вывода. Для адресации может использоваться содержимое регистра, например:

M2 DX; M2 ES:125A; M1 DS:SI.

M2 FE00:0 (информация о Bios)

M2 FFFF:5

G {Стартовый адрес} {, Адрес останова}. Команда предназначена для выполнения программы или ее части в автоматическом режиме, начиная со стартового адреса (start addr). По умолчанию используется сегментный регистр CS. Если указан адрес останова (stop addr), то происходит остановка программы при его достижении. Прервать выполнение программы в автоматическом режиме можно, нажимая клавиши Ctrl–Esc. Если при выполнении программы встретилась точка останова break_addr, то автоматическое выполнение программы прекращается. Если стартовый адрес не указан, то программа выполняется с текущего адреса определяемого CS:IP.

QUIT {R{ESIDENT}}. Команда предназначена для завершения работы AFD. Единственный параметр {R} позволяет оставить ее резидентной в памяти. Когда программа находится резидентно в памяти, обращение к ней выполняется нажатием комбинации клавиш Ctrl-Esc.

A {Адрес}. Команда предназначена для перехода в поле программы и позволяет изменять ее текст, набирая команды в мнемониках Ассемблера. Если не указывается адрес, то курсор устанавливается в текущую команду, на которую указывает программный счетчик CS:IP. Нажатие на клавишу Enter вызывает трансляцию набранной или измененной команды. Выход из режима набор/редактирование осуществляется нажатием на клавиши Ctrl– Enter. Раздвижки команд не происходит, т. е. новые команды записываются поверх старых, т.к. каждая команда «привязана» к конкретному месту в памяти. Необходимо также иметь в виду, что разные команды имеют разную длину, поэтому если необходимо зарезервировать место для будущих команд можно воспользоваться однобайтовой командой означающей пустую операцию – NOP (код 90). При этой операции не выполняется никаких действий.

P Адрес, Строка.

Команда записывает в память по адресу первого операнда строку информации. По умолчанию используется сегментный регистр CS. Строку можно задавать в символьном или в цифровом виде, например:

P 1000,”12345678”, P 1000,11 22 33 44 55 66 77 88

По смещению 1000 в оперативной памяти изменяется 8 байт.

F Адрес, Повторение, Строка.

Команда предназначена для записи в память строки информации с определенным коэффициентом повторения. Операнд «Повторение» позволяет указать количество записываемых в память копий строки. По умолчанию при адресации используется сегментный регистр DS. . Например:

♦ F 200,5,1 2 3 4 5

По смещению 200 в память занесется 25 байт:

01 02 03 04 05 01 02 03 04 05 …

♦ F 500,3,“alfa”

По смещению 500 в память заносится 12 байт:

61 6C 66 61 61 6C 66 61 61 6C 66 61

PD Адрес, Длина {, Имя_файла}. Данная команда предназначена для печати машинного кода программы в мнемониках языка Ассемблер. Печать идет на принтер или в указанный файл. По умолчанию используется сегментный регистр CS. Операнд Длина указывает на количество распечатываемых байт, а не инструкций Ассемблера.

PH {/A} Адрес, Длина {, Имя_файла}. Команда аналогична PD и используется для печати машинного кода программы в виде

шестнадцатиричного дампа. Параметр /A ограничивает вывод данных ASCII- таблицы диапазоном от 20h до 7Fh. Например, при отработке инструкций:

PD 100,100, progr.cod PH 100,100, progr.hex выводится 256 Байт по адресу CS:100 в файл текущего каталога – progr.cod в дизассемблерном виде и в файл progr.hex – в виде шестнадцатиричного дампа.

? = выражение. Данная команда является встроенным целочисленным калькулятором. Ее удобно использовать для расчета различных констант, адресов и переменных. В выражение справа от знака равенства могут входить как шестнадцатеричные, так и десятичные значения, которые начинаются знаком %. Результатом является шестнадцатеричное или десятичное значение без знака выводимое строкой красного цвета. Например:

? = %256 получим >Result =0100 ? = AA-BB получим >Result =0011

? = AA*BB получим >Result =7C2E ? %=35*5 - %10 получим >Result =%255.

Пример 1

Создать файл длиной 256 байтов. Половина файла содержит код 37h, остальная – 67h. Первый байт в файле 00H, последний – FFh. Записать файл в корневой каталог на диск C: под именем PRIMER1.dat.

Решение:

Для записи файла необходимо сформировать область памяти согласно условию задачи. Эту область памяти назовем буфером и выберем адрес этого буфера в текущем кодовом сегменте – 1000h.

Для решения задачи необходимо уметь оперировать с числами в шестнадцатиричной форме. Число 256 в шестнадцатеричном виде – 100h, а половина длины файла

256/2=128 = 80h. Для контроля начало первой половины буфера памяти отобразим в окне Memory 1 командой:

M1 1000

Начало второй половины буфера памяти отобразим во 2 окне Memory 2 командой:

M2 1080

Заполним первую и вторую половину буфера:

F 1000,80,37

F 1080,80,67

Записываем байт в начало и конец буфера:

F 1000,1,0

F 10FF,1,FF

Сохраняем буфер в виде файла: W 1000,100,c:\primer2.dat

На диске c:\ создался файл длиной 256 байт, который можно просмотреть в виде шестнадцатеричного дампа памяти, например встроенным редактором в оболочках Norton Commander или Far manager.

Задачи для самостоятельного выполнения

Задача 1

Создать файл длиной 1001 байт, половина которого заполнена строкой «Miha», а вторая половина строкой «Rita». В 501 байте должен находиться символ "+" Записать файл на диск под именем polovina.dat. Просмотрите полученный файл текстовым редактором в символьном и шестнадцатеричном виде.

Рекомендации:

Для создания буфера в памяти можно воспользоваться командами отладчика F, P, W.

Разобраться с решением примера 1 (п.4).

Не используйте область PSP для своего буфера данных. Область PSP – это первые 256 байт текущего сегмента.

Задача 2

Составить файл на диске с именем data.bin. Файл должен содержать 256 байт, байты идут в порядке возрастания:

00 01 02 03 04 05 ... FF.

Данные для файла data.bin готовит ваша программа с именем data.com. Программа пишется в формате COM с адреса 100h.

Рекомендации. Использовать адресацию к памяти внутри одного сегмента. Составить цикл с количеством повторений 256, внутри которого меняется адрес байта буфера (регистр SI) и содержимое байта буфера (регистр AL).

Тема 01.01.05 Основные машинные команды Ассемблер
Практические занятия. Работа и использование отладчика AFDP.
№4. Команды передачи данных. Арифметические команды.
№5. Логические операторы и команды сдвига
№6. Команды передачи управления. Команды цикла

Цель работы:

Получить навыки работы с основными командами отладчика

Структура машинной команды

Машинная команда представляет собой закодированное по определенным правилам указание микропроцессору на выполнение некоторой операции или действия. Длина машинной команды может составлять целое число байт: 1, 2, 3, 4..., например:

	Команда	*Мнемоника*	*операнд(ы)*
Однобайтовая	50	**PUSH**	AX;
Двухбайтовая	CD 21	**INT**	21;

Трехбайтовая	05 00 10	**ADD**	AX, 1000;
Четырехбайтовая	2B 06 00 10	**SUB**	AX, [1000].

Каждая команда содержит элементы, определяющие: • *код операции*, то есть какое действие необходимо сделать (это обязательный элемент);

• *операнды*, то есть объекты, над которыми необходимо что-то сделать;

• *типы операндов* (обычно задаются неявно исходя из команды или операндов).

Способы задания операндов команды

♦ *Операнды могут задаваться неявно на микропрограммном уровне*. В этом случае команда не содержит явных операндов, и они используются по умолчанию. Например, команды **STC** и **CLC** воздействует на флаг **CF** (устанавливает его в «1» или в «0»).

♦ *Операнд может задаваться непосредственно в самой команде.* Непосредственный операнд может быть только источником. Например,

MOV AX, 1122; **SUB** [BX], 5F.

♦ *Операнд может находиться в одном из* 8/16/32 битных *регистров.* Например: **INC** CH – увеличивает регистр CH на 1; **ADD** BX, AX – суммируется содержимое регистров AX и BX и результат заносится в BX.

♦ *Операнд может находиться в памяти.* Это наиболее сложный и в тоже время наиболее гибкий и чаще всего используемый способ задания операндов. Более подробно способы адресации к памяти рассмотрены в п. 2.2.

♦ *Операндом является порт ввода-вывода.* Источником информации или приемником выступает аккумулятор AL, AX, EAX. Выбор регистра определяется разрядностью порта.

Номер порта задается в регистре DX.

♦ *Операнд находится в стеке.* В этом случае команда содержит один операнд в виде шестнадцатибитного регистра, константы или адреса памяти.

Адресация к памяти

При программировании на языке машинных команд различают следующие основные способы адресации к памяти: – **абсолютная прямая**, например: **MOV [200], AL** записать в ячейку памяти DS:200 содержимое регистра AL;

– **относительная прямая**, например: **JC** 50 – перейти на метку 50, если флаг CF=1. Несмотря на то, что метка указана явно, на самом деле при компиляции в команду входит смещение в формате SHORT (целое однобайтовое со знаком) относительно адреса следующей команды;

– **косвенная адресация.** Различают следующие ее разновидности:

косвенная базовая со смещением, косвенная индексная со смещением, косвенная базовая индексная адресация со смещением и т.п.

При адресации ячеек памяти передача данных в память и из памяти может происходить с использованием сегментных регистров **SR** (DS, SS, ES, CS, FS, GS), базовых регистров **BR** (BX, BP) и индексных регистров **IR** (SI, DI). Общий вид косвенной адресации к памяти для процессоров в реальном режиме их работы можно записать следующим образом:

SR: [BR+IR+(D8 или D16 или D32)],

где D8, D16; D32 – восьми, шестнадцати или тридцатидвухразрядная константа;

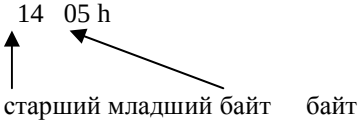
SR – любой из сегментных регистров;

BR – базовый регистр (BP, BX); IR – индексный регистр (SI, DI).

В квадратных скобках указано смещение относительно сегментного регистра SR. Любой из трех компонентов смещения является необязательным, но наличие хотя бы одного из них необходимо. Недопустимо одновременное использование двух базовых или двух индексных регистров.

Сегментный регистр является также необязательным параметром, и если он не указан явно, то действует следующее соглашение: если в качестве базового регистра используется регистр BP, то подразумевается сегментный регистр SS, во всех остальных случаях – регистр DS.

Процессор посредством различных команд адресации к памяти обеспечивает доступ к отдельным байтам, к словам (2 смежным Байтам), к двойным словам (4 Байтам) или к четырем словам памяти (8 Байтам). Рассмотрим, как хранятся числа в памяти, например, целое двухбайтовое число без знака 5125 = 1405h. Понятно, что числа в памяти компьютера хранятся только в двоичном виде. Для удобства человека-пользователя данные числа в отладчике или языке Ассемблер представляются в более компактном шестнадцатеричном виде.



После выполнения следующих машинных команд:

MOV AX, 1405

MOV [1025], AX в оперативной памяти будет следующая картина:

(ячейка) (ячейка)

Числа как бы переворачиваются по адресам. Необходимо всегда помнить, что младшая часть числа всегда хранится в младшем адресе.

ОСНОВНЫЕ МАШИННЫЕ КОМАНДЫ

В данном разделе приведен необходимый минимальный набор и назначение простейших команд. Для удобства практического применения и отражения специфики команды их принято делить на группы: передачи данных, арифметические, логические, цепочечные и т.д. Команда представляет собой некое число в двоичном виде. Человеку довольно трудно оперировать с последовательностями нулей и единиц и он использует их символьные модели – мнемоники.

С точки зрения процессора, нет принципиальной разницы между данными и командами. Данные и машинные команды находятся в одном пространстве памяти в виде последовательности нулей и единиц. Процессор, исполняя содержимое некоторых последовательных ячеек памяти, всегда пытается трактовать его как коды машинной команды, а если это не так, то происходит аварийное завершение программы, содержащей некорректный фрагмент. Надо четко себе представлять, где находятся данные, а где программа.

Смысл многих команд и алгоритм их работы далеко неочевиден, как может показаться на первый взгляд. Некоторые из них имеют свойства, которыми можно воспользоваться в ситуациях, когда команда применяется не по прямому назначению. Тех, кого заинтересует более глубокое понимание работы приведенных ниже машинных команд и алгоритм функционирования других команд можно рекомендовать учебники по ассемблеру [3 - 6].

1. Команды передачи данных

MOV <операнд назначение>, <операнд источник> Это основная команда пересылки данных. Она реализует самые разнообразные варианты пересылки. XCHG < операнд 1>, < операнд 2>

Переставляет или меняет содержимое операндов между собой. Операнды обоих этих команд должны быть согласованы по разрядности. Есть ряд особенностей применения этих команд.

Команда не может передавать данные между двумя адресами памяти;

Нельзя загрузить в сегментный регистр константу или данное из памяти;

Нельзя переслать данное из одного сегментного регистра в другой.

Команда PUSH <источник> - сохраняет значение слова в стеке для последующего использования. Регистр SP указывает на текущее слово в вершине стека (указатель стека). Команда PUSH автоматически умень-

шает значение в регистре SP на 2 и передает слово из указанного операнда в новую вершину стека по адресу

SS:SP.

Например:

```
        PUSH    AX
        PUSH    CX
  PUSH  CS
  PUSH  1234 и т. п.
```

Команда POP <назначение> передает слово, помещенное ранее в стек, в указанный операнд. Регистр SP указывает на текущее слово в вершине стека. Команда POP извлекает слово из стека и увеличивает значение в регистре SP на 2. Например:

```
 PUSH DS  POP   ES
```

При этом SP увеличивается на 2. После выполнения этих команд сегментный регистр ES примет значение DS.

2. Арифметические команды

ADD < операнд1 >, < операнд2 > Команда сложения со следующим принципом действия:

операнд1= операнд1+операнд2 ADC < операнд1 >, < операнд2 >

Команда сложения с учетом флага переноса CF: операнд1= операнд1+операнд2+значение CF

Данные команды воздействуют на флаги AF, CF, OF, PF, SF и ZF. INC <операнд >

Операция инкремента, т.е. увеличения значения операнда на 1.

Команда воздействует на флаги AF, OF, PF, SF и ZF.

SUB < операнд1 >, < операнд2 >

Команда вычитания; ее принцип действия: операнд1= операнд1 – операнд2

Команда воздействует на флаги AF, CF, OF, PF, SF и ZF. SBB < операнд1 >, < операнд2 >

Вычитание с учетом заема (флага CF) операнд1= операнд1-операнд2 – значение CF

DEC <назначение>

Операция декремента, т.е. уменьшает содержимое операнда на 1.

Команда воздействует на флаги AF, OF, PF, SF и ZF.

MUL <сомножитель1 >

Беззнаковое умножение имеет один операнд (сомножитель 1). Второй операнд (сомножитель 2) задается неявно. Так как в общем случае результат умножения больше, чем любой из сомножителей, то его размер и местоположение должны быть тоже определены однозначно. Варианты размеров сомножителей и размещение второго операнда и результата приведены в табл.1.

При операции MUL левый единичный бит рассматривается как бит данных, а не как знаковый бит. Команда воздействует на флаги CF и OF.
DIV <делитель >

Выполняет целочисленное деление чисел без знака. Местоположение делимого фиксировано, как и в команде умножения, зависит от размера делителя. Результатом команды деления являются значения частного и остатка. Варианты местоположения и размеров операндов операции деления показаны в табл.2.

Таблица 1

Расположение операндов и результата при умножении

Сомножитель 1	Сомножитель 2	Результат
Байт (8 бит)	AL	16 бит в AX: AL –младшая часть результата; AH – старшая часть результата
Слово (16 бит)	AX	32 бит в паре DX;AX: причем AX – младшая часть; DX – старшая часть
Двойное слово (32 бита)	EAX	64 бит в паре EDX;EAX: EAX – младшая часть, EDX –старшая часть

Таблица 2

Расположение операндов и результата при делении

Делимое	Делитель	Частное	Остаток
AX	Однобайтовый регистр или ячейка памяти размером 1 Б	AL	AH
DX – старшая часть; AX – младшая часть	Двухбайтовый регистр или ячейка памяти размером 2 Б	AX	DX
EDX – старшая часть; EAX – младшая часть	Двойное слово 4 Б, регистр или ячейка памяти	EAX	EDX

Левый единичный бит рассматривается как бит данных, а не как знаковый бит. После выполнения команды деления содержимое флагов неопределенно, но возможно возникновения немаскируемого аппаратного прерывания с номером 0, называемого «деление на ноль».

3.Логические операторы и команды сдвига

Три логических операции – AND (логическое И), OR (логическое ИЛИ), XOR (исключающее ИЛИ) дают возможность манипулировать отдельными битами в двоичных величинах. Можно устанавливать или сбрасывать единичные биты без влияния на другие, выделять из байта или слова один или группу битов и другие операции. Эти команды имеют два операнда, первый из которых является источником и приемником.

Команда AND часто применяется для маскирования (выделения) битов в байтах и словах. Например, выделить младший полубайт, находящийся в регистре AH:

AND AH, 0F или в двоичном виде: AND AH, 00001111b

Четыре старшие двоичные цифры будут установлены в «0» независимо оттого, что там было.

Логическая команда OR используется для изменений отдельных битов, не влияя при этом, на другие биты. Например, установить в регистре AX первый и 15 бит в «1»: OR AX, 8001 или в двоичном виде

OR AX, 1000000000000001b

Логическая команда XOR является удобным инструментом для переключения отдельных битов из состояния «off» в «on» и наоборот. Если оба бита имеют одинаковое значение, то результат операции XOR равен 0, в противном случае – 1. Это используется для наиболее быстрого обнуления данных регистра, например:

XOR SI, SI XOR AL, AL

Поскольку маска XOR изменяет на противоположный каждый бит начального значения, то повторное применение той же маски к результату восстанавливает первоначальное значение битов. Это широко используется в коммуникационных сетях, шифровании данных и программ.

Другой важной операцией над двоичными значениями является сдвиг битов влево и вправо и ротации через флаг переноса CF. Команды делятся на четыре группы:

♦ Простые сдвиги SHL, SHR;

♦ Арифметические сдвиги SAL, SAR; ♦ Простые ротации ROL, ROR; ♦ Ротации через флаг CF – RCR, RCL.

Общий вид этих команд следующий: КОМАНДА <операнд >,<колич. сдвигаемых бит>.

Выполняют сдвиг всех битов операнда влево или вправо. Сдвиги могут выполняться для однобайтового или двухбайтового операнда, находящегося в регистре или в памяти. Сдвиг на один бит кодируется в

мнемонике константой – «1». Сдвиг более чем на один бит требует указания регистра CL, который содержит счетчик сдвига.

Команды SAL и SHL сдвигают биты влево определенное число раз, и правый, освобождающийся бит, заполняют нулевым значением. Команда SHR сдвигает биты вправо определенное число раз, и левый, освобождающийся бит, заполняет нулевым значением. Команда SAR сдвигает биты вправо определенное число раз, и левый, освобождающийся бит, всегда заполняется исходным значением знакового бита. Особенность работы циклических сдвигов ROL и ROR в том, что «уходящий» бит не теряется, а возвращается, но с другого конца. Во всех случаях значения битов, выдвигаемых за разрядную сетку, помещаются во флаг переноса CF.

4. Команды передачи управления

Команда безусловного перехода: JMP <адрес>

Она выполняет переход по указанному адресу при любых условиях.

Команды условного перехода передают управление на основе анализа некоторых условий или данных, имеют следующий вид:

Jxx <адрес>

Осуществляется переход по указанному адресу при выполнении условия заданного мнемоникой команды. Если заданное условие не выполняется, переход не осуществляется, а выполняется команда, следующая по порядку. Перед командой условного перехода обычно ставится команда: CMP <операнд1 >, <операнд2>

Она сравнивает два операнда. По своему действию эта команда аналогична команде SUB, но в отличие от нее не меняет содержимого операндов, но также воздействует на регистр флагов.

Все команды условного перехода действуют в зависимости от содержимого одного или нескольких флагов и перечислены вместе в табл.3. Например: Сравнить два байта по адресу 1000h и

1001h и если значения в них одинаковые, то перейти на метку 150h.

```
MOV     AL, [1000]
CMP     [1001], AL
JE      150  ; перейти на метку 150,
; если содержимое AL равно байту памяти
; по адресу ds:1001
```

Отличие команд «выше – ниже» и «больше – меньше» заключается в том, что первые команды выражают отношение между операндами без знака, а вторые со знаком.

Таблица 3

Команды условного перехода

Инструкция	Переход если	Анализируемые флаги
JA	Выше	(cf=0) и (zf=0)
JAE	Выше или равно	(cf=0)
JB	Ниже	(cf=1)
JBE	Ниже или равно	(cf=1) или (zf=1)
JE	Равно	(zf=1)
JNE	Не равно	(zf=0)
JG	Больше	(sf=of) и (zf=0)
JGE	Больше или равно	(sf=of)
JL	Меньше	(sf<>of)
JLE	Меньше или равно	(sf<>of) или (zf=1)

5. Команды цикла

Управляют выполнением группы команд определенное число раз, например:

LOOP <метка перехода >

До начала выполнения цикла в регистр CX должно быть загружено число выполняемых повторов. Команда LOOP находится в конце цикла, где она уменьшает значение в регистре CX на единицу. Если значение в регистре CX не равно нулю, то команда передает управление по адресу, указанному в операнде. В противном случае управление передается на следующую после LOOP команду.

Команды LOOPZ и LOOPNZ – расширяют действие команды LOOP тем, что дополнительно анализируют флаг ZF, что дает возможность организовать досрочный выход из цикла, используя этот флаг в качестве индикатора. Так цикл LOOPZ прекращается либо по нулевому значению CX=0, либо, если флаг ZF=0 (результат ненулевой операции).

Пример 2

Набрать простейшую программу в отладчике по смещению 100, имеющую линейный алгоритм. Выполнить ее по шагам и записать в тетради изменения регистров, флагов и стека.

```
0100 B81111      MOV   AX,1111   ;AX=1111
0103 BB3322      MOV   BX,2233   ;BX=2233
0106 48          DEC   AX        ;AX=1110
0107 43          INC   BX        ;BX=2234
0108 01D8        ADD   AX,BX     ;AX=3344
010A 50          PUSH  AX        ;SP=FFFC
010B 29C0        SUB   AX,AX     ;AX=0; ZF=1
```

```
010D F7D0       NOT     AX          ;AX=FFFF
010F 050500     ADD     AX,0005  ;AX=4
0112 F7D8       NEG     AX          ;AX=-4=FFFC
0114 29D8       SUB     AX,BX           ;AX=DDC8
0116 93         XCHG    AX,BX       ;AX⇔ BX
0117 5E         POP     SI              ;SI=3344; SP=FFFE
0118 86C4       XCHG    AL,AH       ;AL⇔ AH; AX=3422
011A F7D8       NEG     AX          ;AX=CBDE
011C F8         CLC                 ;CF=0
011D F9         STC                 ;CF=1
011E 90         NOP
```

Пример 3

Переслать в пределах одного сегмента из области памяти с адресом NAME1 в область с адресом NAME2 десять байт информации. В качестве примера взять NAME1=0200, NAME2=0300.

Решение:

В отладчике отсутствует возможность написание комментариев, но мы их будем писать для пояснения логики работы через точку запятой как это принято в языке Assembler.

```
0100 BE0002     MOV SI,0200     ; инициализация адресов NAME1 и
0103 BF0003     MOV DI,0300     ; NAME2
0106 B90A00     MOV             ; счетчик пересылки 10 байт
                CX,000A
0109 8A04       MOV AL,[SI]     ; переслать байт из NAME1
010B 8805       MOV [DI],AL     ; переслать байт в NAME2
010D 46         INC  SI         ; увеличить SI на 1
010E 47         INC  DI         ; увеличить DI на 1
010F E2F8       LOOP 0109       ; уменьшить CX на 1,
                                ; если CX≠ 0 уйти на 0109
0111 C3         RET             ; вернуться в родительскую программу
```

После набора каждой строки нажимается клавиша Enter и встроенный в отладчик интерпретатор анализирует ее и вставляет в память соответствующие машинные коды. Строка MOV SI,0200 генерирует 3 байта машинного кода BE 00 02; строка MOV AL,[SI] генерирует 2 байта машинного кода 8A04. Если строка не соответствует синтаксису мнемоник машинных команд, то выдается сообщение об ошибке.

В отладчике можно использовать символьные метки и ссылки на них. В этом примере адрес 0109 можно пометить как m1, тогда машинный код данного примера будет выглядеть следующим образом:

```
0100 BE0002        MOV
SI,0200 0103 BF0003
MOV    DI,0300 0106 B90A00
MOV    CX,000A
  m1:                ;метка или адрес
0109 8A04          MOV   AL,[SI]
010B 8805          MOV   [DI],AL
010D 46            INC
SI 010E 47           INC
DI
010F E2F8          LOOP  m1
0111 C3          RET
```

Пример 4

По адресу A1 находится массив из четырех однобайтовых переменных. Найти сумму элементов массива и записать по адресу A1 – 8.

Решение:

```
0100   BE0010          MOV SI, 1000  ; задаем A1
                       =1000h
0103   56              PUSH SI
0104   BB0000          MOV BX, 0000
0107   8A1C            MOV BL, [SI]
0109   B90300          MOV CX, 0003
010C   46              INC  SI
; формируем сумму в регистре BX с учетом переноса
010D   021C            ADD BL, [SI]
010F   80D700          ADC BH, 00
0112   E2F8            LOOP 010C
0114   5E              POP SI
0115   83EE08          SUB SI, 0008 ;A1-8 адрес для сум-
                       мы
0118   891C            MOV [SI], BX
011A   90              NOP
```

До выполнения:

2684:0FF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ……. 2684:1000 **FF FF 11 22** 00 00 00 00 00 00 00 00 00 00 00 ."…… *После выполнения:*

2684:0FF0 00 00 00 00 00 00 00 00 **31 02** 00 00 00 00 00 00 ……1……. 2684:1000 **FF FF 11 22** 00 00 00 00 00 00 00 00 00 00 00 ."……

Полученное значение суммы 231.

Пример 5

В области памяти, заданной начальными и конечными адресами A1 и A2 (A2>A1), отсортировать однобайтовые данные в порядке их возрастания.

Решение:

Будем использовать простую сортировку массива a[n] методом пузырька. На языке Pascal данный алгоритм сортировки имеет следующий вид: n:=A2-A1; {Массив имеет n+1 элемент, так как нумеруется с 0}

```
For i:=0   to n-2 do
  For j=i+1 to n-1 do
  If a[i]<a[j] then
Begin
          tmp:=a[i];
          a[i]:=a[j];
          a[j]:=tmp;
end;
```

В машинных кодах данная программа приведена ниже (занимает в памяти 37h=55 Байт); в качестве аналогичных переменных языка Паскаль используются следующие регистры: i – CX, j – BX, tmp – ячейка с адресом DS:0137.

```
0100  BE 39 01          MOV SI, 0139       ; заносим адрес
                                           ;A1=139 – начала данных
0103  33 DB             XOR BX, BX         ; обнуление BX
0105  33 C9             XOR CX, CX         ; обнуление CX
0107  33 D2             XOR DX, DX         ; обнуление DX
; вычисляем размер массива, зная A2=159
; и занесение полученного значения в ячейку памяти размером
; в одно слово
0109  C70637015901      MOV [0137], 0159
010F  29 36 37 01       SUB [0137], SI

0113  8B 16 37 01       MOV  DX, [137]

0117  4A                DEC      DX

0118  8B D9              MOV  BX,CX  ; начало основного цикла сортировки
011A  8A 20              MOV  AH, [SI+BX]  ; берем очередной элемент
```

```
011C  43                INC     BX  ; и начинаем проверять
; его с оставшимся массивом
011D  8A 00       MOV   AL,[SI+BX]  ; заносим
; следующий элемент из массива
011F  3A C4       CMP   AL, AH ; сравниваем элементы
0121  73 0A       JNC             012D  ; проверка (Jae=Jnb)
; если «левый» элемент не больше «правого», то берем
; новый элемент, иначе меняем их местами
0123  86 C4             XCHG      AL, AH
0125  88 00             MOV       [SI+BX],
                                  AL
0127  87 D9             XCHG      BX, CX
0129  88 20             MOV       [SI+BX],
                                  AH
012B  87 D9             XCHG      BX, CX
012D  3B DA             CMP       BX, DX
; если не достигнут конец массива, то берем новый
; элемент, иначе повторяем для следующего элемента
012F  72 EB             JB        011C
0131  41                INC       CX
0132  3B CA             CMP       CX,DX
0134  72 E2             JB      0118
0136  C3                RET
```

Пример 6

Составить программу ekran.com, которая меняет все атрибуты текстового экрана таким образом, чтобы можно было писать красным цветом на светло-синем фоне. Использовать прямой доступ к видеобуферу.

Решение:

Определим необходимые константы:

общее количество символов на экране - 2000 = 7D0h;

устанавливаемый атрибут символов - 1001 0100b = 94h

цвет цвет фона текста

```
100  B8 00 B8          MOV  AX, B800  ; базовый сегментный
; адрес видеобуфера
103  8E C0       MOV   ES, AX
105  B9 D0 07    MOV   CX,    7D0    ;    CX    =2000   количество
; атрибутов в видеобуфере
108  BB 01 00    MOV   BX, 1     ; смещение первого
```

```
; атрибута относительно начала видеобуфера
10B  B0 94       MOV    AL, 94 ; AL= 10010100b – код атрибута
;          цикл замены атрибутов
10D  26 88 07    MOV    ES:[BX], AL
110  43  INC     BX 111  43       INC      BX
112  E3 F9       LOOP   010D
114  C3 RET              ; выход в OS
```

Данная программа занимает в памяти (длина программы) 115h-100h=15h=21 байт.

Задачи для самостоятельного выполнения

Задача 3

Решить задачу 1, без использования команды F. Заполнение буфера памяти данными производит созданная вами программа.

Задача 4

Составить программу для подсчета двухбайтовой контрольной суммы области памяти внутри одного сегмента DS от адреса A1 до A2.

Рекомендации Адреса A1, A2 - двухбайтовые. Начальный адрес A1 поместите в регистр SI, конечный A2 – в регистр DI. В качестве примера возьмите A1=1000h, A2=10FFh. Для проверки работы программы обнулите эту область и запишите туда несколько байт, чтобы их можно для проверки было легко сложить.

Задача 5

Составить программу, которая копирует свой код в другую область памяти в том же кодовом сегменте, например, сразу же за текстом программы. Можно ли создать программу, которая копирует свой код N раз?

Тема 01.01.05 Написание программ на языке Ассемблер

Практическое занятие №7. Написание программ с использованием циклических конструкций.

Цель работы:

Целью работы является закрепление знаний по командам условного и безусловного переходов и циклов на примере программ на языке ассемблера, а также приобретение навыков написания программ с циклами.

Основная информация

Алгоритмическая структура “цикл”, как известно, обеспечивает выполнение некоторой последовательности действий, которая называется телом цикла.

Выделяется три типа циклов: цикл “ДЛЯ”, цикл “ПОКА”, цикл “ДО”. Друг от друга различные типы циклов отличаются в основном лишь способом проверки окончания цикла.

Типы циклов

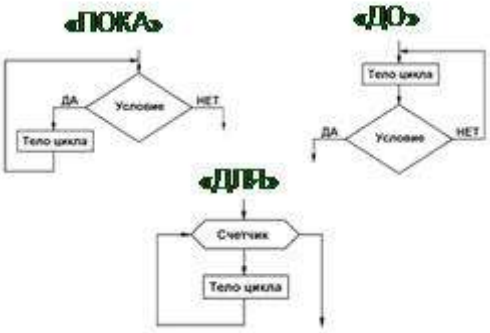
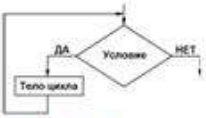


Рис. 1

В языке программирования Паскаль для реализации каждого типа цикла имеются специальные операторы, но любой из этих трех типов можно организовать при помощи условного оператора и оператора безусловного перехода.

Цикл «ПОКА»

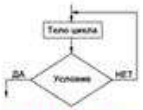


While X > 0 do S

```
A:  If X > 0 then begin
    S; тело цикла
    Goto A;
    end;
```

Рис. 2

Цикл «ДО»



Repeat S until X > 0

```
A:  S; тело цикла
    If (X < 0) or (X = 0) then Goto A;
```

Рис. 3

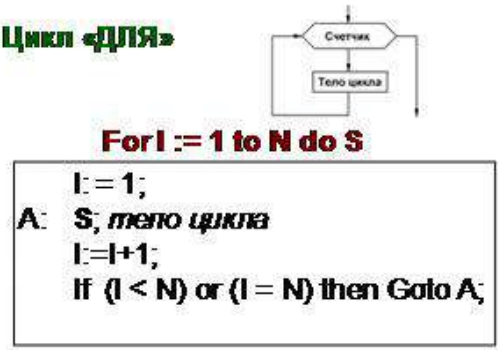


Рис. 4

Система команд языка Ассемблер тоже позволяет организовать циклическое выполнение некоторого фрагмента программы, к примеру, используя команды условной передачи управления или команду безусловного перехода JMP.

Рис. 5

Рис. 6

Рис. 7

Как и в языке Паскаль, в Ассемблере существует специальная команда, которая позволяет сокращать листинг циклической программы.

Это команда LOOP <метка>.

Данная команда выполняет следующие функции:

Автоматически уменьшает значение счетчика.

Выполняет проверку на выход из цикла.

Выполняет переход на начало тела цикла.

Команда LOOP может быть использована лишь в случае цикла с известным числом повторений, т.е. цикла “ДЛЯ”. Количество повторений цикла должно быть присвоено регистру CX до начала цикла.

Рис. 8

Таким образом, команда LOOP заменила тройку команд:

DEC CX

CMP CX , 0

JNE A2

Рассмотрим использование этой команды на практике.

Пример: Составим программу, которая выводит на экран 1000 нулей.

```
(1) prg segment para public 'code'
(2) assume cs:prg,ss:prg,es:prg,ds:prg
(3) org 100h
(4) start: jmp go
(5) go:
(6) mov ax, 0600h
(7) mov bh,07
(8) mov cx, 0000
(9) mov dx,184fh
(10) mov cx,1000
(11) Zero:
(12) mov ah,02
(13) mov dl,30h
(14) int 21h
(15) loop Zero
(16) ret
(17) prg ends
(18) end start
```

Строки с (1) по (10) и с (16) по (18) вы уже знаете.

Строка (11) – это метка (начало цикла). Строка (15) – конец цикла. Все, что находится в пределах строк (11) – (15), является циклом. Сам цикл будет повторяться 1000 раз, для чего мы и заносим в CX число 1000 (строка (10)).

В строке (12) заносим в регистр ah число 02 (запрос функции вывода одного символа).

В строке (13) в регистр dl заносим код выводимого символа (код символа “0” – 30h).

В строке (14) вызываем прерывание int 21h.

Теперь на экране появится первый ноль. Остается уменьшить счетчик (CX) на 1 и повторить. Что мы и делаем в строке (15).

Задания:

Задача 1: Составить фрагмент программы на языке Ассемблер, подсчитывающий сумму первых 10 натуральных чисел (результат записать в AX).

Решение:

```
…..
mov cx,10
mov ax,00
summa:
add ax,cx
loop summa
…..
```

Задача 2: Составить фрагмент программы на языке Ассемблер, вычисляющий значение выражения: $\sum_{x=1}^{5} 2x$ (результат записать в AX).

Решение:

…..

```
mov BX,00
mov CX,05
sum:
mov AX,02
mul CX
add BX,AX
loop sum
```

…..

Задача 3: Составить фрагмент программы на языке Ассемблер, вычисляющий факториал заданного числа К (К – от 0 до 8).

Решение:

…..

```
mov ax, 1
mov cx, K
F: mul cx
loop F
```

…..

Тема 01.01.05 Написание программ на языке Ассемблер

Практическое занятие №8. Написание программ с использованием подпрограмм.

Цель работы:

Получить навыки создания процедур с использованием подпрограмм

Основная информация

Процедуры могут получать или не получать параметры из вызывающей процедуры и могут возвращать или не возвращать результаты. Процедуры, которые что-либо возвращают, называют функциями (Паскаль).

Различают два основных механизма передачи параметров процедуре:

- по значению;
- по ссылке.

Пусть procedure - имя процедуры, a value - имя некоторой области памяти (переменная). Тогда вызов процедуры с передачей параметра по значению выглядит следующим образом:

```
mov  AX, word ptr value ; процедура получает копию
call procedure; входного параметра
```

Вызов с передачей параметра по ссылке будет выглядеть так:

```
mov  AX, offset value ; процедура получает адрес
call procedure ; входного или выходного параметра
```

Для обмена информацией между вызывающим кодом и процедурой могут использоваться такие каналы:

- регистры;
- стек;
- глобальные переменные;
- поток кода;
- блок параметров.

Передача параметров в регистрах

Если число параметров и сами параметры невелики, лучшим метсом для их передачи являются регистры. Примерами могут служить вызовы функций DOS и BIOS. Языки высокого уровня обычно используют регистр AX (EAX) для размещения результата, который возвращается функцией.

Передача параметров через стек

Параметры помещаются в стек перед вызовом процедуры. Именно такой метод используют языки высокого уровня (C, Pascal). Для чтения параметров из стека применяют не команду POP, а регистр BP, в который помещают адрес вершины стека после входа в процедуру:

```
push  параметр1; поместить параметр в стек
push  параметр2
call  procedure
add   SP,  4 ; освободить стек от параметров
.....
procedure  PROC  near
push  BP
mov   BP,  SP
 (команды, которые могут использовать стек)
mov   AX,  [BP+4] ; считать параметр2. Его адрес [BP+4], потому что ; при выполнении команды call в стек поместили адрес возврата (2 байта для
; процедуры типа NEAR), а потом еще и BP - 2 байта.
mov   BX,  [BP+6]      ; считать параметр1.
 (остальные команды)
pop   BP
ret
procedure  ENDP
```

При использовании стека для передачи параметров процедуры возникает два вопроса: кто должен удалять параметры из стека (процедура или вызывающий код) и в каком порядке помещать параметры в стек. Все возможные варианты ответа на этот вопрос имеют свои "за" и "против". Так, например, если стек освобождает процедура (команда ret число_байтов), то код программы получается меньшим, а если за освобождение стека от параметров отвечает вызывающая функция, то становится возможным последовательными командами CALL вызвать несколько функций с одними и теми же параметрами. Первый способ - более строгий (Pascal), второй - дает больше возможностей для оптимизации (C). Кроме того, в языке C параметры помещают в стек в обратном порядке (справа налево), так что становятся возможными функции с изменяемым числом параметров.

Передача параметров в потоке кода

Передаваемые данные размещаются прямо в коде программы, сразу после команды call, например (так реализовна процедура print в одной из стандартных библиотек процедур для ассемблера UCRLIB):

```
call print
DB  "This ASCIZ-line will be printed", 0
(следующая команда)
```

Чтобы прочитать параметр, процедура должна использовать его адрес, который автоматически передается в стеке как адрес возврата из процедуры. При этом адрес возврата должен быть изменен на первый байт после конца блока данных перед выполнением команды RET.

Передача параметров в потоке кода (так же как и передача параметров в стеке в обратном порядке) позволяет передавать различное число параметров или один параметр различной длины.

Передача параметров в блоке параметров

Блок параметров - участок памяти, содержащий параметры и располагающийся обычно в сегменте данных. Процедура получает адрес начала такого блока. Таким методом реализованы многие функции DOS и BIOS (поиск файла - использует блок параметров DTA, загрузка и исполнение программы - используется блок параметров EPB).

Локальные переменные

Наиболее распространенный способ хранения локальных переменных - стек. Принято располагать локальные переменные сразу после сохраненного значения регистра BP, так что на них можно ссылаться как [BP-2], [BP-4], [BP-6] и т.д.

Рассмотрим процедуру examp, в которой используются параметры x, y, z и локальные переменные a, b, c (все эти величины предполагаются 2-х байтовыми):

```
examp    proc   near
x  equ [BP+8] ; параметры процедуры
y equ  [BP+6]
z equ [BP+4]
a equ  [BP- 2]  ; локальные переменные
b  equ [BP- 4]
cequ  [BP- 6]
pushBP ; сохранить BP
mov BP,  SP   ; установить BP для себя
sub  SP,  6 ; зарезервировать 6 байт для локальных переменных
 (тело процедуры)
mov  SP,  BP   ; выбросить из стека локальные переменные
pop BP  ; восстановить BP вызвавшей процедуры
ret 6 ; возврат с удалением трех параметров из стека
examp  endp
```

При вызове процедуры examp стек будет выглядеть следующим образом:

```
 [BP+8]  =   x
 [BP+6]  =   y
 [BP+4]  =   z
 [BP+2]  =   IP
BP®     [BP+0]  =   BP
 [BP- 2]  =   a
 [BP- 4]  =   b
SP®      [BP- 6]  =   c
```

В связи с тем, что последовательности команд

```
push      BP
mov       BP,  SP
sub       SP,  6
```

и

```
mov       SP,  BP
pop       BP
```

используются очень часто при построении процедур, были введены специальные команды ENTER и LEAVE, которые их заменяют. С использованием этих команд процедура examp будет выглядеть следующим образом:

```
examp    proc   near
x          equ       [BP+8]
y          equ       [BP+6]
z           equ       [BP+4]
a        equ         [BP- 2]
```

```
b       equ         [BP- 4]
c       equ         [BP- 6]
enter   6,  0                   ; второй параметр - уровень вложенности процедуры
 (тело процедуры)
leave
ret     6
examp   endp
```

Активизационной записью процедуры называется следующая информация, размещаемая в стеке: параметры процедуры, IP (адрес возврата), BP (предыдущее значение).

Стековый кадр - область стека, отведенная для активизационной записи и локальных переменных процедуры.

Тема 01.01.05 Написание программ на языке Ассемблер

Практическое занятие№9. Написание программ с использованием макросов.

Цель работы: изучить возможности оптимизации программы, написанной на языке Assembler.

Альтернативы секции .const

Главной альтернативой секции .const является простое объявление символьной константы.

```
  CONST_VALUE = 678h
  .data
  Dd CONST_VALUE
  .code
...more code...
  Mov edi, CONST_VALUE
  Xor eax, CONST_VALUE
```

Вторая не менее важная альтернатива – это определение "макросимвола" (я это сам придумал, а, по-научному, абсолютный символ или "прозвище"). Он задаётся через директиву equ.

примеры

```
  CONST1 equ 0123h
  CONST2 equ 14d*15h
  CONST3 equ "slovo"
  CONST4 equ 56-45
  CONST5 equ (offset metka1)
  CONST6 equ (offset metka2+offset metka3)
  CONST7 equ (CONST2+10b)
  CONST8 equ CONST7/2
```

```
CONST9 equ (offset metka1-offse metka5)
CONST10 equ (offset metka4+CONST4)
CONST11 equ add edx,edi
```

... и так далее

Директива equ используется в там же, где определяются символьные константы. Это ещё не все возможности этой директивы. Можно дать прозвище некоторой команде, например:

Command1 equ mov eax, esi.

После этого при каждом упоминании command1 будет подразумеваться команда move eax, esi.

Пример

```
Mov eax, CONST1
Add edi, CONST2
Xor ebp, CONST7
CONST11
Sub edx, CONST8
```

Объявлять equ надо в начале файла там же где объявляются структуры и константы.

Макросы.

Макрос - это набор команд. С помощью equ можно создавать "прозвище" только для одной команды, а с помощью макросов можно создавать "прозвище" для нескольких команд. Для создания макроса надо использовать директивы macro и соответственно endm.

```
Firstmacro macro
                Sub ebp, esp
                Mov eax, ebp
          Endm
```

Теперь если компилятор встречает слово Firstmacro, то он автоматически заменяет его на тело макроса. Также макросу можно передавать параметры.

```
Secondmacro macro param1, param2
                        Add edi, param1
                        Sub esi, param2
              endm
.code    ;  использование
.....
Secondmacro 55h,edx
```

У макросов очень много возможностей.

Включаемые файлы.

Иногда надоедает (особенно при создании оконных приложений) объявлять в каждой программе одни и те же структуры и константы. Хотелось бы один раз их задать, а потом их использовать в

каждой своей программе как в Си или Паскале. Для этого предназначены включаемые файлы *.inc. В них можно задать структуры, константы, макросы. Для включения такого файла надо использовать директиву include. После этой директивы надо указать путь к включаемому файлу абсолютный или относительный. При написании такого файла просто думайте, что вы находитесь после директивы .model и до .data.

Пример:

Файл sample.asm

```
;=======[CUT HERE]========
.386
.model  flat, stdcall

include sample.inc

extrn MessageBoxA:PROC

.data

Msg         db "First ASSEMBLER program",0h
Ttl         db 'Hello, World!!!!',0h
RCTNGL RECT ?
.code

start:
  call MessageBoxA,0,offset Msg,offset Ttl,MB_OKCANCEL
  call ExitProcess, 0
end     start
;=====[CUT HERE]==========
```

Файл sample.inc

```
;=======[CUT HERE]========
extrn ExitProcess:PROC

UINT    EQU <dd>  ; 32 bits for WIN32

RECT    struc
      rcLeft        UINT ?
      rcTop         UINT ?
      rcRight       UINT ?
      rcBottom      UINT ?
RECT    ends
```

```
MB_OK                = 0000H
MB_OKCANCEL          = 0001H
MB_ABORTRETRYIGNORE  = 0002H
MB_YESNOCANCEL       = 0003H
MB_YESNO             = 0004H
MB_RETRYCANCEL       = 0005H
;=====[CUT HERE]===========
```

Директива include заменяется на всё содержимое включаемого файла.

Упрощённый вызов API функций в TASM

Если в директиве .model укажем модель вызова функций, то вызывать API будет намного проще

Call <функция>, <параметр1>,<параметр2>,<параметр3>

Теперь первая программа будет иметь такой вид:

```
  .386
.model  flat, stdcall

extrn MessageBoxA:PROC
extrn ExitProcess:PROC
.data

Msg         db "First ASSEMBLER program",0h
Ttl         db 'Hello, World!!!!',0h


.code

start:
  call MessageBoxA,0,offset Msg,offset Ttl,0
  call ExitProcess, 0
end     start
```

Команда call func, param1,pram2, paramn при компиляции автоматически преобразуется в

```
Push paramn
Push param2
Push param1
Call func
```

Тема 01.01.06 Управление системными ресурсами компьютера

Практическое занятие №10. Работа с памятью и клавиатурой

Цель работы:

Организация и модели памяти, адресация

Память – способность объекта обеспечивать хранение данных.

Все объекты, над которыми выполняются команды, как и сами команды, хранятся в памяти компьютера.

Память состоит из ячеек, в каждой из которых содержится 1 ***бит*** информации, принимающий одно из двух значений: 0 или 1. Биты обрабатывают группами фиксированного размера. Для этого группы бит могут записываться и считываться за одну базовую операцию. Группа из 8 бит называется **.**

Байты последовательно располагаются в памяти компьютера.

- 1 килобайт (Кбайт) = 2^{10} = 1 024 байт
- 1 мегабайт (Мбайт) = 2^{10} Кбайт = 2^{20} байт = 1 048 576 байт
- 1 гигабайт (Гбайт) = 2^{10} Мбайт = 2^{30} байт = 1 073 741 824 байт

Для доступа к памяти с целью записи или чтения отдельных элементов информации используются ***идентификаторы***, определяющие их расположение в памяти. Каждому идентификатору в соответствие ставится ***адрес***. В качестве адресов используются числа из диапазона от 0 до 2^k-1 со значением k, достаточным для адресации всей памяти компьютера .Все 2^k адресов составляют ***адресное пространство компьютера***.

Способы адресации байтов

Существует прямой и обратный способы адресации байтов.

При ***обратном*** способе адресации байты адресуются слева направо, так что самый старший (левый) байт слова имеет наименьший адрес.

Прямым способом называется противоположная система адресации. Компиляторы высокоуровневых языков поддерживают прямой способ адресации.

Объект занимает целое слово. Поэтому для того, чтобы обратиться к нему в памяти, нужно указать адрес, по которому этот объект хранится.

Организация памяти

Физическая память, к которой микропроцессор имеет доступ по шине адреса, называется ***оперативной памятью*** ОП (или оперативным запоминающим устройством — ОЗУ).

Механизм управления памятью полностью аппаратный, т.е. программа сама не может сформировать физический адрес памяти на адресной шине.
Микропроцессор аппаратно поддерживает несколько моделей использования оперативной памяти:

- сегментированную модель
- страничную модель
- плоскую модель

В сегментированной модели память для программы делится на непрерывные области памяти, называемые ***сегментами***. Программа может обращаться только к данным, которые находятся в этих сегментах.

Сегмент представляет собой независимый, поддерживаемый на аппаратном уровне блок памяти.

Сегментация — механизм адресации, обеспечивающий существование нескольких независимых адресных пространств как в пределах одной задачи, так и в системе в целом для защиты задач от взаимного влияния.

Каждая программа в общем случае может состоять из любого количества сегментов, но непосредственный доступ она имеет только к 3 основным сегментам и к 3 дополнительным сегментам, обслуживаемых 6 сегментными регистрами. К основным сегментам относятся:

- Сегмент кодов (.CODE) – содержит машинные команды для выполнения. Обычно первая выполняемая команда находится в начале этого сегмента, и операционная система передает управление по адресу данного сегмента для выполнения программы. Регистр сегмента кодов (CS) адресует данный сегмент.
- Сегмент данных (.DATA) – содержит определенные данные, константы и рабочие области, необходимые программе. Регистр сегмента данных (DS) адресует данный сегмент.
- Сегмент стека (.STACK). Стек содержит адреса возврата как для программы (для возврата в операционную систему), так и для вызовов подпрограмм (для возврата в главную программу). Регистр сегмента стека (SS) адресует данный сегмент. Адрес текущей вершины стека задается регистрами SS:ESP.

Регистры дополнительных сегментов (ES, FS, GS), предназначены для специального использования.

Для доступа к данным внутри сегмента обращение производится относительно начала сегмента линейно, т.е. начиная с 0 и заканчивая адресом, равным размеру сегмента. Для обращения к любому адресу в программе, компьютер складывает адрес в регистре сегмента и ***смещение*** — расположение требуемого адреса относительно начала сегмента. Например, первый байт в сегменте кодов имеет смещение 0, второй байт – 1 и так далее.

Таким образом, для обращения к конкретному физическому адресу ОЗУ необходимо определить адрес начала сегмента и смещение внутри сегмента.

Физический адрес принято записывать парой этих значений, разделенных двоеточием

сегмент : смещение

Страничная модель памяти – это надстройка над сегментной моделью. ОЗУ делится на блоки фиксированного размера, кратные степени 2, например 4 Кб. Каждый такой блок называется ***страницей***. Основное достоинство страничного способа распределения памяти — минимально возможная фрагментация. Однако такая организация памяти не использует память достаточно эффективно за счет фиксированного размера страниц.

Плоская модель памяти предполагает, что задача состоит из одного сегмента, который, в свою очередь, разбит на страницы. Достоинства:

- при использовании плоской модели памяти упрощается создание и операционной системы, и систем программирования;
- уменьшаются расходы памяти на поддержку системных информационных структур.

В абсолютном большинстве современных 32(64)-разрядных операционных систем (для микропроцессоров Intel) используется плоская модель памяти.

Модели памяти

Директива .MODEL определяет модель памяти, используемую программой. После этой директивы в программе находятся директивы объявления сегментов (.DATA, .STACK, .CODE, SEGMENT). Синтаксис задания модели памяти

.MODEL модификатор **МодельПамяти** СоглашениеОВызовах

Параметр **МодельПамяти** является обязательным.

Основные модели памяти:

Модель памяти	**Адресация кода**	**Адресация данных**	**Операционная система**	**Чередование кода и данных**
TINY	NEAR	NEAR	MS-DOS	Допустимо
SMALL	NEAR	NEAR	MS-DOS, Windows	Нет
MEDIUM	FAR	NEAR	MS-DOS, Windows	Нет
COMPACT	NEAR	FAR	MS-DOS, Windows	Нет
LARGE	FAR	FAR	MS-DOS, Windows	Нет
HUGE	FAR	FAR	MS-DOS, Windows	Нет
FLAT	NEAR	NEAR	Windows NT, Windows 2000, Windows XP,	Допустимо

			Windows Vista	

Модель tiny работает только в 16-разрядных приложениях MS-DOS. В этой модели все данные и код располагаются в одном физическом сегменте. Размер программного файла в этом случае не превышает 64 Кбайт.

Модель small поддерживает один сегмент кода и один сегмент данных. Данные и код при использовании этой модели адресуются как near (ближние).

Модель medium поддерживает несколько сегментов программного кода и один сегмент данных, при этом все ссылки в сегментах программного кода по умолчанию считаются дальними (far), а ссылки в сегменте данных — ближними (near).

Модель compact поддерживает несколько сегментов данных, в которых используется дальняя адресация данных (far), и один сегмент кода с ближней адресацией (near).

Модель large поддерживает несколько сегментов кода и несколько сегментов данных. По умолчанию все ссылки на код и данные считаются дальними (far).

Модель huge практически эквивалентна модели памяти large.

Особого внимания заслуживает модель памяти flat, которая используется только в 32-разрядных операционных системах. В ней данные и код размещены в одном 32-разрядном сегменте. Для использования в программе модели flat перед директивой .model flat следует разместить одну из директив:

.386
.486
.586
.686

Желательно указывать тот тип процессора, который используется в машине, хотя это не является обязательным требованием. Операционная система автоматически инициализирует сегментные регистры при загрузке программы, поэтому модифицировать их нужно только в случае если требуется смешивать в одной программе 16-разрядный и 32-разрядный код. Адресация данных и кода является ближней (near), при этом все адреса и указатели являются 32-разрядными.

Параметр **модификатор** используется для определения типов сегментов и может принимать значения use16 (сегменты выбранной модели используются как 16-битные) или use32 (сегменты выбранной модели используются как 32-битные).

Параметр **СоглашениеОВызовах** используется для определения способа передачи параметров при вызове процедуры из других

языков, в том числе и языков высокого уровня (C++, Pascal). Параметр может принимать следующие значения:

C,
BASIC,
FORTRAN,
PASCAL,
SYSCALL,
STDCALL.

При разработке модулей на ассемблере, которые будут применяться в программах, написанных на языках высокого уровня, обращайте внимание на то, какие соглашения о вызовах поддерживает тот или иной язык. Используются при анализе интерфейса программ на ассемблере с программами на языках высокого уровня.

Ассемблерная программа, например, получает число имеющихся дисковых накопителей следующим образом. При этом, нельзя забывать о восстановлении первоначального значения в порте B.

```
IN   AL,61H          ;получаем значение из порта B
OR   AL,10000000B    ;устанавливаем бит 7 в 1
OUT  61H,AL          ;заменяем байт
IN   AL,60H          ;получаем значение из порта A
MOV  CL,6            ;подготовка для сдвига AL
SHR  AL,CL           ;сдвигаем 2 старших бита на 6 позиций
INC  AL              ;начинаем счет с 1, а не с 0
MOV  NUM_DRIVES,AL   ;получаем число накопителей
IN   AL,61H          ;подготовка к восстановлению порта B
AND  AL,01111111B    ;сбрасываем бит 7
OUT  61H,AL          ;восстанавливаем байт
```

Определение числа и типа периферийных устройств.

При старте компьютера ROM-BIOS проверяет присоединенное оборудование, сообщая о результатах своей проверки в регистр статуса. Этот регистр занимает два байта, начиная с 0040:0010. Ниже приведены значения битов относятся ко всем машинам, пока не оговорено обратное:

бит 0 если 1, то присутствует НГМД
1 XT,AT:1 = есть мат. сопроцессор (PC,PCjr: не использ.)
2-3 11 = базовая память 64K (AT: не используется)
4-5 Активный видеоадаптер
6-7 число НГМД (если бит 0 = 1)
8 PCjr:0 = есть DMA (PC,XT,AT: не используется)

9-11 число адаптеров коммуникации
12 1 = есть игровой порт (AT: не используется)
13 PCjr :есть серийный принтер (PC,XT,AT: не использ.)
14-15 число присоединенных принтеров

Число портов коммуникации может быть получено из области данных BIOS. BIOS отводит четыре 2-байтных поля для хранения базовых адресов вплоть до четырех COM портов. Базовый адрес – это младший из адресов портов, относящихся к группе портов, имеющих доступ к данному каналу коммуникации. Эти четыре поля начинаются с адреса 0040:0008. Порту COM1 соответствует адрес :0008, а COM2 - 000A. Если это поле содержит 0, то соответствующий порт отсутствует. Таким образом, если слово по адресу: 0008 отлично от нуля, а по адресу 000A - нулевое, то имеется один порт коммуникации.

Ревизия количества памяти.

Информация о количестве банков памяти может быть прочитана через порт B микросхемы интерфейса с периферией 8255. BIOS хранит двухбайтную переменную по адресу 0040:0013, которая сообщает число килобайт используемой
памяти. Для PCjr бит 3 порта 62H (порт C микросхемы 8255) равен нулю, когда машина имеет добавочные 64K памяти. AT дает особо полную информацию о памяти. Регистры 15H (младший) и 16H (старший) микросхемы информации о конфигурации говорят сколько памяти установлено на системной плате. Память канала ввода/вывода для AT сообщается регистрами 17H и 18H (с инкрементом 512K). Память сверх 1 мегабайта доступна через регистры 30H и 31H (также с инкрементом 512K, вплоть до 15 мегабайт). Если AT имеет 128K на плате расширения, то установлен бит 7 регистра 33. Во всех случаях надо сначала послать номер регистра в порт 70H, а затем прочитать значение из порта 71H.

Тема 01.01.06 Управление системными ресурсами компьютера
Практическое занятие №11. Работа с таймером и генерация звука
Цель работы:
Изучить возможности работы с таймером и звуком.

Встроенный динамик – это системное устройство, поэтому в отличие от музыкальных сопроцессоров не требует никаких драйверов и доступно всегда. В архитектуре IBM PC существуют две возможности управлять «спикером»:

с помощью первого бита порта 61h;
при программировании второго канала таймера Intel 8253

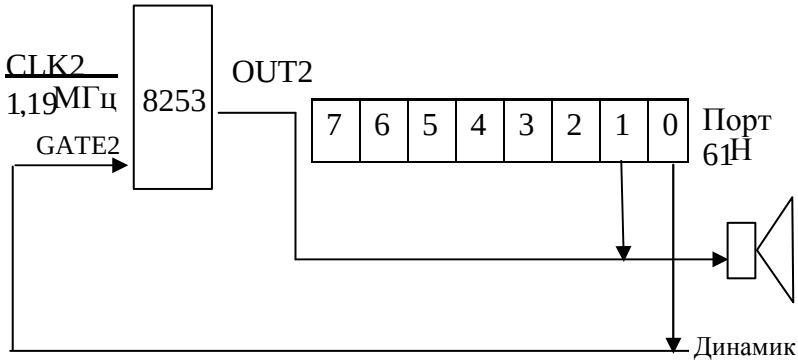


Рис. 1. Получение звука через PC Speaker

Комбинируя эти возможности, можно получить специальные звуковые эффекты. Аппаратные возможности не позволяют изменить громкость звука, издаваемого динамиками.

Системный порт 61h работает как на чтение, так и на запись.

Ворота GATE2 закрываются записью «0» в нулевой бит 61h порта.

Это позволяет производить ряд манипуляций со звуком. Первый бит порта 61h связан с PC speaker и также может быть использован для генерации звука. Спикер в отличие от звуковой платы является системным устройством, а не драйверным, и присутствует в любой компьютерной системе, поэтому он широко используется для сигнализации различных событий, программных ошибок и аппаратных сбоев.

Генерация звука с помощью порта 61h

Метод состоит в периодическом включении и выключении с желаемой частотой первого бита порта 61h. Если переключать значение бита с максимально возможной частотой, то мы вряд ли что услышим, так как получаемая частота будет довольно высокой.

Слышимые человеком звуки ограничены диапазоном от 20 до 20000 Гц. Звуки, воспроизводимые SPEAKER, имеют еще более узкий диапазон 50–12000 Гц. Между переключениями необходимо вставлять задержки по времени τвкл, τвыкл. (рис. 4.5). Нулевой бит порта 61h управляет воротами канала 2 таймера, который связан с динамиком. Этот бит сбрасываем в 0, чтобы таймер не влиял на получаемый звук.

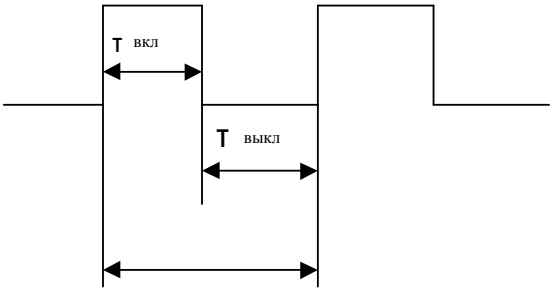


Рис. 2. Формирование прямоугольных импульсов, подаваемых на системный динамик: период получаемых колебаний T=τвкл +τвыкл; полученная частота звука f=1/T Гц

Задача 6

Составить программу для генерации звука посредством порта 61h.

Задержки по времени осуществляются программно, поэтому, чем производительнее процессор, тем выше будет тон получаемого звука и меньше времени он будет продолжаться. Фрагмент этой программы можно использовать как простейший тест быстродействия компьютера.

Решение:

```
 ; Основной фрагмент программы zvuk_prt.asm
cli                         ; запрет аппаратных прерываний
mov         DX, 1000        ; длительность тонов в периодах
in          AL, 61h   ; получаем значение порта 61H
and         AL, 1111111Ob ; отключение динамика от
metka1:     8253
or          AL, 00000010b  ; включим динамик
out         61h, AL
; задержка на половину периода
MOV     CX, 300h metka2:
LOOP metka2
  AND   AL, 11111101b           ; выключение динамика
  OUT   61h, AL
; задержка на половину периода
MOV     CX, 300h metka3:
LOOP metka3
  DEC   DX
  JNZ   metka1
```

```
  STI                    ; разрешить аппаратные прерывания
```

Генерация звука посредством таймера

Более предпочтительным является способ получения звука посредством таймера, т.к. процессор «не участвует» в получении самого звука и может заниматься другой работой.

Задача 7

Получить звук с частотой f=440 Гц. Звук прекратить нажатием любой клавиши на клавиатуре.

Решение:

```
  ;Основной фрагмент программы zvuk_t.asm
; Биты 0 и 1 порта 61h предварительно необходимо установить в «1»

in          AL, 61h     ; читаем порт 61H
or          AL, 00000011b   ; установление двух     ; младших битов в «1»
out         61h, AL
mov         AL, 10110110b       ; управляющий регистр для 8253
out         43h, al
mov         AX, A91h            ; 1190000/440=2705=A91h
out         42h, AL     ; посылаем младший байт
mov         AL, AH
out         42h, AL     ; посылаем старший байт

 ; слышим звук с частотой 440 Гц

  mov     AX, 0C08h         ; очистка буфера,
  ; ввод символа без ЭХА на экран
int         21h                 ; ждем нажатие клавиши in      AL, 61h
  and       AL, 11111100b   ; сбрасываем два младших бита
  out       61h, AL
; звук выключается
```

Тема 01.01.06 Управление системными ресурсами компьютера

Практическое занятие №12. Резидентные программы

Цель работы:

Изучить основы работы с резидентными программами.

Перехват прерываний и создание резидентных программ

Перечень вариантов заданий к практической работе (по вариантам).

1. Через временной интервал (например, 1 минута) на экран выводится какое-либо сообщение. Через 10-20 секунд сообщение снимается и работа ПЭВМ продолжается обычным образом.

2. При нажатии любой клавиши на экран выдается просьба нажать эту клавишу еще раз. При повторном нажатии просьба с экрана снимается и работа ПЭВМ продолжается обычным образом.

3. При нажатии клавиши ENTER в центр экрана выводится сообщение: "Отдыхаю, подождите минутку". Через 10-20 секунд сообщение снимается и работа ПЭВМ продолжается обычным образом.

4. ПЭВМ реагирует на клавишу "стрелка-вверх" как на клавишу "стрелка-вниз" (и наоборот), на клавишу "стрелка-влево" как на клавишу "стрелка-вправо" (и наоборот).

5. При нажатии клавиши F1 программа очищает экран и безостановочно выводит на экран сообщение "Не хочу вам помогать!", прокручивая при этом экран вверх. Секунд через 10-20 этот процесс прекращается, восстанавливается экран и работа ПЭВМ продолжается обычным образом.

6. Через равные промежутки времени (например 30 секунд) резидент блокирует/разблокирует клавиатуру.

7. При нажатии клавиши T сообщается текущее системное время.

8. Нажатие клавиши D замедляет/восстанавливает реакцию системы на нажатие клавиш клавиатуры. Замедление реакции должно быть достаточным для того чтобы его можно было заметить визуально.

9. При нажатии клавиши R очищается правая, а при нажатии L-левая половина экрана. Через 10-20 секунд после нажатия любой из этих клавиш экран восстанавливается.

10. Нажатие клавиши G производит переключение видеоадаптера между текстовым и графическим режимом.

11. Нажатие клавиши I инвертирует цвета экрана.

12. После установки резидента система перестает реагировать на нажатие клавиши F7. На другие клавиши система реагирует обычным образом.

Тема 01.01.06 Управление системными ресурсами компьютера

Практическое занятие №13. Формирование видеоизображения в различных режимах

Цель работы:

Изучить методы управления видео контроллеров

Стандартные типы видеоадаптеров

Устройство, которое называется видеоадаптером (или видеоплатой, видеокартой), есть в каждом компьютере. В виде устройства, интегрированного в системную плату, либо в качестве самостоятельного ком-

понента - платы расширения. **Главная функция, выполняемая видеокартой, это преобразование полученной от центрального процессора информации и команд в формат, который воспринимается электроникой монитора, для создания изображения на экране.** Монитор обычно является неотъемлемой частью любой системы, с помощью которого пользователь получает визуальную информацию.

MDA (Monochrome Display Adapter - монохромный адаптер дисплея) -простейший видеоадаптер, применявшийся в первых IBM PC. Работает в текстовом режиме с разрешением 80x25 (720x350, матрица символа - 9x14), поддерживает пять атрибутов текста: обычный, яркий, инверсный, подчеркнутый и мигающий. Частота строчной развертки - 15 Кгц. Интерфейс с монитором - цифровой: сигналы синхронизации, основной видеосигнал, дополнительный сигнал яркости.

HGC (Hercules Graphics Card - графическая карта Hercules) - расширение MDA с графическим режимом 720x348, разработанное фирмой Hercules.

CGA (Color Graphics Adapter - цветной графический адаптер) - первый адаптер с графическими возможностями. Работает либо в текстовом режиме с разрешениями 40x25 и 80x25 (матрица символа - 8x8), либо в графическом с разрешениями 320x200 или 640x200. В текстовых режимах доступно 256 атрибутов символа - 16 цветов символа и 16 цветов фона (либо 8 цветов фона и атрибут мигания), в графических режимах доступно четыре палитры по четыре цвета каждая в режиме 320x200, режим 640x200 - монохромный. Вывод информации на экран требовал синхронизации с разверткой, в противном случае возникали конфликты по видеопамяти, проявляющиеся в виде "снега" на экране. Частота строчной развертки - 15 Кгц. Интерфейс с монитором - цифровой: сигналы синхронизации, основной видеосигнал (три канала - красный, зеленый, синий), дополнительный сигнал яркости.

EGA (Enhanced Graphics Adapter - улучшенный графический адаптер) -дальнейшее развитие CGA, примененное в первых PC AT. Добавлено разрешение 640x350, что в текстовых режимах дает формат 80x25 при матрице символа 8x14 и 80x43 - при матрице 8x8. Количество одновременно отображаемых цветов - по-прежнему 16, однако палитра расширена до 64 цветов (по два разряда яркости на каждый цвет). Введен промежуточный буфер для передаваемого на монитор потока данных, благодаря чему отпала необходимость в синхронизации при выводе в текстовых режимах, структура видеопамяти сделана на основе так называемых битовых плоскостей - "слоев", каждый из которых в графическом режиме содержит биты только своего цвета, а в текстовых режимах по плоскостям разделяются собственно текст и данные знакогенератора. Совместим с MDA и CGA. Частоты строчной развертки - 15 и 18 Кгц. Интерфейс

с монитором - цифровой: сигналы синхронизации, видеосигнал (по две линии на каждый из основных цветов).

MCGA (Multicolor Graphics Adapter - многоцветный графический адаптер) -введен фирмой IBM в ранних моделях PS/2. Добавлено разрешение 640x400 (текст), что дает формат 80x25 при матрице символа 8x16 и 80x50 - при матрице 8x8. Количество воспроизводимых цветов увеличено до 262144 (по 64 уровня на каждый из основных цветов). Помимо палитры, введено понятие таблицы цветов, через которую выполняется преобразование 64-цветного пространства цветов EGA в пространство цветов MCGA. Введен также видеорежим 320x200x256, в котором вместо битовых плоскостей используется представление экрана непрерывной областью памяти объемом 64000 байт, где каждый байт описывает цвет соответствующей ему точки экрана. Совместим с CGA по всем режимам, а с EGA - по текстовым, за исключением размера матрицы символа. Частота строчной развертки -31 Кгц, для эмуляции режимов CGA используется так называемое двойное сканирование - дублирование каждой строки формата Nx200 в режиме Nx400. интерфейс с монитором - аналогово-цифровой: цифровые сигналы синхронизации, аналоговые сигналы основных цветов, передаваемые монитору без дискретизации. Поддерживает подключение монохромного монитора и его автоматическое опознание - при этом в видео-BIOS включается режим суммирования цветов по так называемой шкале серого (grayscale) для получения полутонового черно-белого изображения. Суммирование выполняется только при выводе через BIOS - при непосредственной записи в видеопамять на монитор попадает только сигнал зеленого цвета (если он не имеет встроенного цветосмесителя).

VGA (Video Graphics A rray - массив визуальной графики) - расширение MCGA, совместимое с EGA, введен фирмой IBM в средних моделях PS/2. Фактический стандарт видеоадаптера с конца 80-х годов. Добавлен текстовый режим 720x400 для эмуляции MDA и графический режим 640x480 с доступом через битовые плоскости. В режиме 640x480 используется так называемая квадратная точка (соотношение количества точек по горизонтали и вертикали совпадает со стандартным соотношением сторон экрана - 4:3). Совместим с MDA, CGA и EGA, интерфейс с монитором идентичен MCGA.

IBM 8514/a - специализированный адаптер для работы с высокими разрешениями (640x480x256 и 1024x768x256), с элементами графического ускорителя. Не поддерживает видеорежимы VGA. интерфейс с монитором аналогичен VGA/MCGA.

IBM XGA - следующий специализированный адаптер IBM. расширено цветовое пространство (режим 640x480x64к), добавлен текстовый

режим 132x25 (1056x400). Интерфейс с монитором аналогичен VGA/MCGA.

SVGA (Super VGA - "сверх" VGA) - расширение VGA с добавлением более высоких разрешений и дополнительного сервиса. Видеорежимы добавляются из ряда 800x600, 1024x768, 1152x864, 1280x1024, 1600x1200 - все с соотношением 4:3. Цветовое пространство расширено до 65536 (High Color) или 16.7 млн. (True Color). Также добавляются расширенные текстовые режимы формата 132x25, 132x43, 132x50. Из дополнительного сервиса добавлена поддержка VBE.

2.2. Видеопамять

Один из компонентов компьютера, от которого требуется наибольшая производительность, это графический контроллер, являющийся сердцем всех мультимедиа систем. Фраза требуется производительность означает, что некоторые вещи происходят настолько быстро, насколько это обеспечивается пропускной способностью. Пропускная способность обычно измеряется в мегабайтах в секунду и показывает скорость, с которой происходит обмен данными между видеопамятью и графическим контроллером.

На производительность графической подсистемы влияют несколько факторов:

- скорость центрального процессора (CPU)
- скорость интерфейсной шины (PCI или AGP)
- скорость видеопамяти
- скорость графического контроллера

Видеокарта состоит из четырех основных устройств: памяти, контроллера, ЦАП и ПЗУ.

Видеопамять служит для хранения изображения. От ее объема зависит максимально возможное полное разрешение видеокарты - AxBxC, где A - количество точек по горизонтали, B - по вертикали, и C - количество возможных цветов каждой точки. Например, для разрешения 640x480x16 достаточно 256 Кб, для 800x600x256 -512 Кб, для 1024x768x65536 (другое обозначение - 1024x768x64к) - 2 Мб, и т.д. Поскольку для хранения цветов отводится целое число разрядов, количество цветов всегда является степенью двойки (16 цветов - 4 разряда, 256 - 8 разрядов, 64k - 16, и т.д.).

Видеоконтроллер отвечает за вывод изображения из видеопамяти, регенерацию ее содержимого, формирование сигналов развертки для монитора и обработку запросов центрального процессора. Для исключения конфликтов при обращении к памяти со стороны видеоконтроллера и центрального процессора видеоконтроллер имеет отдельный буфер, который в свободное от обращений ЦП время заполняется данными из видеопамяти. Если конфликта избежать не удается - видеоконтроллеру

приходится задерживать обращение ЦП к видеопамяти, что снижает производительность системы; для исключения подобных конфликтов в ряде карт применялась так называемая двухпортовая память, допускающая одновременные обращения со стороны двух устройств.

Многие современные видеоконтроллеры является потоковыми - их работа основана на создании и смешивании воедино нескольких потоков графической информации. Обычно это основное изображение, на которое накладывается изображение аппаратного курсора мыши и отдельное изображение в прямоугольном окне. Видеоконтроллер с потоковой обработкой, а также с аппаратной поддержкой некоторых типовых функций называется акселератором или ускорителем, и служит для разгрузки ЦП от рутинных операций по формированию изображения.

ЦАП (цифроаналоговый преобразователь, DAC) служит для преобразования результирующего потока данных, формируемого видеоконтроллером, в уровни интенсивности цвета, подаваемые на монитор. Все современные мониторы используют аналоговый видеосигнал, поэтому возможный диапазон цветности изображения определяется только параметрами ЦАП. Большинство ЦАП имеют разрядность 8x3 - три канала основных цветов (красный, синий, зеленый, RGB) по 256 уровней яркости на каждый цвет, что в сумме дает 16.7 млн. цветов. Обычно ЦАП совмещен на одном кристалле с видеоконтроллером.

Видео-ПЗУ - постоянное запоминающее устройство, в которое записаны видео-BIOS, экранные шрифты, служебные таблицы и т.п. ПЗУ не используется видеоконтроллером напрямую - к нему обращается только центральный процессор, и в результате выполнения им программ из ПЗУ происходят обращения к видеоконтроллеру и видеопамяти. ПЗУ необходимо только для первоначального запуска адаптера и работы в режиме MS DOS; операционные системы с графическим интерфейсом - Windows или OS/2 - практически не используют ПЗУ для управления адаптером, хотя и могут иметь проблемы в работе при ошибках в программе BIOS, не найденных разработчиками.

На карте обычно размещаются один или несколько разъемов для внутреннего соединения; один из них носит название Feature Connector и служит для предоставления внешним устройствам доступа к видеопамяти и изображению. К этому разъему может подключаться телеприемник, аппаратный декодер MPEG, устройство ввода изображения и т.п. На некоторых картах предусмотрены отдельные разъемы для подобных устройств.

Все возможности видеосистемы компьютера можно реализовать с помощью видеофункций BIOS прерывания int 10h. Прерывание int 10h обеспечивает: смену видеорежима (текстовый или графический); вывод символьной и текстовой информации; смену шрифтов, настройку цвето-

вой палитры, работу с графическим изображением. Программирование видеосистемы с помощью средств BIOS более громоздко, однако большие возможности и высокая скорость вывода обуславливают широкое использование этого метода в прикладных программах.

В данной работе рассматриваются функции BIOS для обслуживания видеосистемы компьютера, а также функции для работы с клавиатурой. Перечислим функции, являющиеся предметом рассмотрения в лабораторной работе.

Int 10h:

- функция 00h - установка видеорежима;
- функция 02h - установка позиции курсора;
- функция 03h - считывание позиции и размера курсора;
- функция 05h - установка видеостраницы;
- функция 06h (07h) - инициализация или прокрутка окна вверх (вниз);
- функция 08h - чтение символа и атрибута в позиции курсора;
- функция 09h - запись символа и атрибута в позицию курсора;
- функция 0Ah - запись символа в позицию курсора с текущим атрибутом;
- функция 0Eh - запись символа в режиме телетайпа с текущим атрибутом;
- функция 0Fh - получить режим дисплея;
- функция 1003h - переключение назначения старшего бита байта атрибута: мерцание/яркость,
- функция 13h - запись строки с заданным атрибутом в режиме телетайпа.

Int 16h:

- функция 00h (10h) - чтение символа с клавиатуры с ожиданием;
- функция 01h(11h)- проверка буфера клавиатуры на наличие в нём символа;
- функция 02h (12h) - получение флагов (расширенной) клавиатуры.

Прямое программирование видеобуфера в текстовом режиме

Современные видеоконтроллеры поддерживают разнообразные текстовые и графические режимы. Текстовые режимы различаются по разрешению (число отображаемых символов по горизонтали и вертикали) и цветовой палитре (монохромный или 16-цветный режим). Для графических режимов основным признаком классификации является коли-

чество одновременно отображаемых цветов и, соответственно, количество бит видеопамяти, отводимое на каждую точку (пиксел) изображения. Различают следующие типы графических режимов:

- монохромный (1-битное кодирование);
- 16-цветный *EGA/VGA* (4-битное кодирование);
- 256-цветный *SVGA* (8-битное кодирование);
- *HiColor* (16-битное кодирование);

- *True Color* (24-битное / 32-битное кодирование).

Всё, что изображено на мониторе - графика, текст - одновременно присутствует в памяти, встроенной в видеоадаптер. Для того чтобы изображение появилось на мониторе, оно должно быть записано в память видеоадаптера.

На экране отображается видеобуфер, соответствующий активной странице. В текстовых режимах для изображения каждого символа отводится 2 байта: байт с ASCII-кодом символа и байт с его атрибутом. Вообще при формировании изображения непосредственно в видеобуфере, в обход программ DOS и BIOS, все управляющие коды ASCII теряют свои управляющие функции и отображаются в виде соответствующих символов. Структура байта атрибутов приведена на рис. 2.1.

Рис. 1.- Структура байта атрибутов

Из рис. 2.1 следует, что каждый символ может принимать любой из 16 возможных цветов, определяемых сочетанием младших 4-х битов. Биты 4-6 байта атрибутов задают цвет фона под данным символом. Последний бит 7, в зависимости от режима видеоадаптера, определяет либо яркость фона под данным символом (тогда фон также может принимать 16 разных цветов), либо мерцание символа (устанавливается DOS по умолчанию).

При загрузке машины устанавливается стандартная палитра, коды цветов которой приведены в табл. 1. Рассмотрим некоторые примеры. Так, в режиме мерцания значение старшего полубайта атрибута 8h обозначает не серый фон, а чёрный при мерцающем символе, цвет которого по-прежнему определяется младшим полубайтом; значение старшего полубайта 0Ch - красный фон при мерцающем символе. Переключение

назначения бита 7 осуществляется подфункцией 03h функции 10h прерывания int 10h.

Таблица 1

Коды цветов стандартной палитры

Код	Цвет	Код	Цвет
0h	Чёрный	8h	Серый
lh	Синий	9h	Голубой
2h	Зелёный	0Ah	Салатовый
3h	Бирюзовый	0Bh	Светло-бирюзовый
4h	Красный	0Ch	Розовый
5h	Фиолетовый	0Dh	Светло-фиолетовый
6h	Коричневый	0Eh	Жёлтый
7h	Белый	0Fh	Ярко- белый

Двухбайтовые коды символов записываются в видеобуфер в том порядке, в каком они должны появиться на экране: первые 80*2 байт соответствуют первой строке экрана, вторые 80*2 байт - второй и т.д. При этом переход на следующую строку экрана определяется не управляющими кодами возврата каретки и перевода строки, а размещением кода в другом месте видеобуфера. Вычислить смещение ячейки в координатах "строка-столбец" (row, clm) можно так:

VidAddr= (row* 160) + (clm* 2)

При большом объёме выводимых данных, информационный кадр формируется заранее в буфере пользователя, располагающегося в сегменте данных программы.

Справочные данные по функциям bios

Регистры общего назначения.

Регистрами общего назначения называются 32-битные регистры EAX, EBX, ECX, EDX, EBP, ESP, ESI и EDI. Данные регистры используются для хранения операндов логических и арифметических команд. Кроме того, они могут использоваться для хранения операндов при вычислении адресов (кроме регистра ESP, который не может быть использован как индексный операнд). Имена указанных регистров наследованы от имен регистров общего назначения процессора 8086 - AX, BX, CX, DX, BP, SP, SI и DI. В Таблице 2 показано, как можно адресовать младшие 16 бит регистров общего назначения процессора i486, используя имена регистров процессора 8086.

Каждый байт 16-битных регистров AX, BX, CX и DX также имеет свое имя. Байты этих регистров называются AH, BH, CH и DH (старшие байты) и AL, BL, CL и DL (младшие байты).

Таблица 2

Имена регистров

8 Бит	16 Бит	32 Бита
AL	AX	EAX
AH	BX	EBX
BL	CX	ECX
BH	DX	EDX
CL	SI	ESI
CH	DI	EDI
DL	BP	EBP
DH	SP	ESP

Прерывание int 10h. Видеофункции BIOS

Функция 00h. Установка видеорежима (табл. 2.2) текущей видеостраницы с очисткой экрана *(быстрая очистка экрана реализуется функцией 06h и 07h, полная очистка экрана 00h).*

Вызов: *AH = 00h,*

Al = видеорежим (код режима задаётся в младших 7 битах, установка в 1 старшего бита запрещает очистку экрана).

Регистры *AX, BP, SI, и DI* – не используются в данной функции.

Функция 02h. Установка позиции курсора.

Задаёт положение курсора на экране в текстовых координатах, с которых в дальнейшем будет выводиться текст. Отсчёт номера строки и столбца ведётся от верхнего левого угла. Курсор можно установить, как в текстовом, так и в графическом режиме, однако, в графическом режиме курсор не виден. BIOS поддерживает до восьми независимых курсоров - по одному на каждую страницу (см. табл. 2.2) независимо от того, какая страница является активной. Функцию *02h* BIOS можно использовать в комбинации с функциями DOS для организации вывода на экран.

Вызов: *AH = 02h; BH* = номер страницы (0,1,...7), **обычно 0;** *DH* = строка; *DL* = столбец.

Регистры *AX, BP, SI* и *DI* – не используются в данной функции.

Функция 03h. Считывание позиции и размера курсора.

Возвращает текущие координаты состояния курсора на выбранной странице. Это даёт возможность временно перейти для работы на другое место экрана, а затем вернуться на старое место. Функцию *03h* BIOS можно использовать в комбинации с функциями DOS для организации вывода на экран.

Вызов: *AH = 03h, BH* = номер страницы (0,1,...7), **обычно 0.**

Возврат: *DH, DL* = строка и столбец текущей позиции курсора, *CH, CL* = первая и последняя строки развёртки курсора – размер курсора.

Регистры AX, BP, SI и DI – не используются в данной функции.

Функция 08h. Чтение символа и атрибута в текущей позиции курсора на выбранной странице.

Вызов: *AH = 08h, (цвет) BH* = номер страницы (0,...,7), **обычно 0.**

Возврат: *AH* = атрибут символа, *AL = ASCII-код* символа.

Регистры *BP, SI* и *D* – не используются в данной функции.

Функция 09h. Запись символа с заданным атрибутом на экран в позицию курсора. Действует как в графическом, так и в текстовом режимах. В графическом режиме символы не должны переходить на следующую строку. *Все коды в AL рассматриваются как символьные и не управляют положением курсора. После вывода символа курсор смещается к следующей позиции функцией 02h.* Коэффициент повторения позволяет выводить строки одинаковых символов. В текстовом режиме символ выводится с указанным в *BL* атрибутом. В графическом - содержимое *BL* влияет только на цвет символа, но не на фон под ним. Графическое изображение под знакоместом затирается.

Вызов: *AH =09h, AL = ASCII-код* символа,

BL — атрибут символа (текстовый режим) или только цвет символа (графический режим),

BH = номер страницы (0,1,...7), *CX*= коэффициент повторения.

Регистры *AX, BP, SI* и *DI* – не используются в данной функции.

Функция 0Ah. Запись символа с текущим атрибутом на экран в позицию курсора. Функция действует как в графическом, так и в текстовом режимах. Символ принимает атрибут, установленный ранее для этой позиции. *Все ASCII-коды в AL рассматриваются как символьные и не управляют положением курсора (также как и в функции 09h). После вывода символа курсор смещается к следующей позиции функцией 02h.*

Вызов: *AH = 0Ah, AL = ASCII-код* символа,

BH = номер страницы (0,1,...7), *CX* = коэффициент повторения.

Регистры *AX, BP, SI* и *DI* – не используются в данной функции.

Функция 0Fh. Получить режим дисплея и номер текущей страницы.

Вызов: *AH = 0Fh.*

Возврат: *AL* = режим дисплея, *AH* = ширина экрана в текстовом формате

BH =номер активной страницы.

Регистры *BP, SI* и *DI* – не используются в данной функции.

Пример. Процедура установки позиции курсора на текущей странице.

Вход: *dh* = строка (0 - 25), *dl* = столбец (0 - 79)

```
Proc SetCursor
;Сохранить регистры (по необходимости)
Mov ah,0Fh
Int 10h
Mov ah,02h
Int l0h
```

;восстановить регистры
Endp SetCursor

Функция 10h. Подфункция *03h*. Переключение бита "мерцание/яркость".

Определяет назначение старшего бита 7 атрибута символа: мерцание символа или повышенная яркость фона.

Вызов: *AX*= 1003h, *BL* = назначение 7-го бита атрибута:

0 - повышенная яркость, 1 - мерцание *(устанавливается по умолчанию)*.

Функция воздействует сразу на все символы экрана, у которых установлен старший бит атрибута фона.

Функция 13h. Запись строки символов с заданными атрибутами.

Записывает строку в текущую страницу видеобуфера, начиная с указанной позиции. Коды *ASCII: 07h* - звонок, *08h* - шаг назад, *0Ah* - перевод строки, *0Dh* - возврат каретки, рассматриваются как управляющие, остальные - как символьные.

Вызов: *AH = I3h, AL* = режим записи:

1. — атрибут символа в *BL,* строка содержит только коды символов, после записи курсор принимает исходное положение (т.е. вывод следующей строки, если не изменить позицию курсора, начинается с изначально установленной позиции);
2. - отличается от режима 0 тем, что после записи курсор остаётся в конце строки;
3. - строка содержит попеременно коды символов и атрибутов (т.е. каждый символ описывается 2 байтами — *ASCII-кодом* и атрибутом), после записи курсор принимает исходное положение;
4. - отличается от режима 2 тем, что по окончании вывода курсор остаётся в конце строки.

BH = номер страницы (0,1,.. .7), BL = атрибут для режимов 0 и 1,

CX - длина символьной строки (в длину входят только коды символов, но не байты атрибутов),

DX = DH.DL — координаты курсора (строка, столбец) в исходной точке вывода строки на экране,

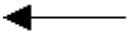
ES:BP = адрес начала строки в памяти.

Пример программы индивидуального задания.

Нарисовать прямоугольник зелёного цвета в любом месте экрана

```
.model small
.stack
100h
VGA_mode equ 13h ; 320x200 256 цветный графический режим
color equ 2 ; цвет линий
x_sise equ 300 ; ширина прямоугольника в пикселах
```

```
y_sise equ 100 ; высота прямоугольника в пикселах
x_pos equ 10 ; положение нижнего левого угла прямоуг-ка
y_pos equ 50
.code
start:
set_mode:
mov ah,00h ; вызов нулевой функции BIOS
mov al,VGA_mode ; и инициализация графического режима
int 10h
set_proc:
mov ah,0Ch ; настройка параметров для вызова функции 0Ch
mov al,color
mov cx,x_pos
mov dx,y_pos
line_1: ;
int 10h
inc cx
cmp cx,(x_pos + x_sise)
jne line_1
line_2: ;
int
int10h
inc dx
cmp dx,(y_pos + y_sise)
jne line_2;
line_3: ;
int 10h
dec cx
cmp cx,x_pos
jne line_3;
line_4: ;
int 10h
dec dx
cmp dx,y_pos
jne line_4
anykey: ; блок отвечающий за завершение приложения
mov ah,1 ; при нажатии любой клавиши
int 16h ; вызов 16h прерывания BIOS, определения
jz anykey ; наличия введенного символа
int 21h
end start ; завершене программы
```



Варианты индивидуального задания:

1. Разработать программу рисования зеленого прямоугольника в любом месте экрана.
2. Разработать программу рисования красного прямоугольного треугольника в нижнем правом углу экрана.
3. Разработать программу рисования синего равнобедренного треугольника в верхнем правом углу экрана.
4. Разработать программу рисования белого ромба в любом месте экрана.
5. Разработать программу рисования желтого квадрата в центре экрана.
6. Разработать программу рисования белой трапеции в любом месте экрана.
7. Разработать программу рисования синего параллелограмма в любом месте экрана.
8. Разработать программу рисования двух любых букв русского алфавита в любом месте экрана.
9. Разработать программу рисования красного закрашенного квадрата в центре экрана.

Раздел 2 МДК.01.02. Прикладное программирование

Тема 01.02.01
Интегрированная среда разработки Microsoft Visual Studio .NET
Практическое занятие № 1,2,3,4
Написание программ с использованием:

1. Редактора кода
2. Иерархической структуры программного кода
3. Контекстного поиска и замены
4. IntelliSense (выпадающий список-подсказка)

Цель работы: Написание программ с использованием интегрированная среда разработки Microsoft Visual Studio .NET
Выполнив работу, Вы будете:
уметь:

- осуществлять разработку кода программного модуля на современных языках программирования;
- выполнять отладку и тестирование программы на уровне модуля;
- использовать инструментальные средства для автоматизации оформления документации;

Технические средства обучения:
персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением
Программное обеспечение:
Embarcadero RAD Studio 2010, 1С: Предприятие 8.3, GIMP, Git, MS Visual Studio 2010
Оборудование лаборатории и рабочих мест лаборатории:

- Пакет прикладных программ «Microsoft Office»: табличный процессор Microsoft Excel, текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer
- Компилятор tasm
- Отладчик машинных кодов AFDP
- MS Visual Studio 2010
- Embarcadero RAD Studio 2010
- 1С: Предприятие 8.3.
- Git.

Реализация рабочей программы ПМ предполагает обязательную учебную и производственную практику.
Оборудование и технологическое оснащение рабочих мест – Полигон учебных баз практики:
персональные компьютеры в локальной сети с доступом к сети Internet – 22 ед. с лицензионным программным и сетевым обеспечением;
локальная сеть с доступом к ресурсам интернет;

сервер

Программное обеспечение:

- Embarcadero RAD Studio 2010,
- 1С: Предприятие 8.3,
- GIMP,
- Git,
- MS Visual Studio 2010.

Задания

Задание 1.

- Установить среду Microsoft Visual Studio
- Создать проект Консольное приложение c#. Для названия проекта используйте следующий синтаксис Lab#_Familia. Пример Lab1_Ivanov. *Лабораторные с неправильно оформленным названием могут быть не приняты или случайно удалены.*
- Вывести на экран две строчки:
  a. В первой строке «Hello World!»
  b. Во второй строке: ФИО и номер группы.
- Удалить из проекта все не нужные блоки using (которые были добавлены средой по умолчанию)
- Добавить в проект строчку содержащую ошибку и попытаться запустить прило-жение
- Исправить ошибку и повторить запуск

Задание 2.

Напишите программу в соответствии с индивидуальным заданием. Программа должна выводить приглашение на ввод данных на русском языке, выводить результат в заданном формате и останавливаться в конце своего выполнения для просмотра результатов ее работы. Создайте новый проект в среде программирования на основе текста вашей программы. Запустите программу, продемонстрируйте результаты ее работы преподавателю.

Вариант 1

С клавиатуры вводится целое число, вывести это число в двоичной, шестнадцатеричной и восьмеричной системе счисления (каждое – в отдельной строке).

Вариант 2

С клавиатуры вводится целое число, вывести на экран квадрат и куб этого числа (каждое значение – в отдельной строке).

Вариант 3

С клавиатуры вводится вещественное число. Вывести значение того числа, округленное до 1, 3 и 5 знаков после запятой.

Вариант 4

С клавиатуры вводится целое число n. Вывести на экран таблицу квадратов и кубов целых чисел от n до n+4, сохраняя выравнивание в столбцах:

4	16		64
5	25	125	
6	36		216

Вариант 5

С клавиатуры вводятся длины катетов прямоугольного треугольника, посчитать и вывести на экран его площадь и длину гипотенузы (каждое значение в отдельной строке).

Порядок выполнения работы:

Составление алгоритма на языке программирования.

Написание программного кода на языке программирования

Отладка программы на ПК

Форма представления результата:

Алгоритм программы

Программный код, составленный на языке программирования

Тема 01.02.02 Разработка консольного приложения

Практическое занятие № 5,6

Написание программ с использованием:

1. Панели инструментов Debug
2. Окна отладки

Цель работы: Написание программ с использованием консольного приложения

Выполнив работу, Вы будете:

уметь:

- осуществлять разработку кода программного модуля на современных языках программирования;
- выполнять отладку и тестирование программы на уровне модуля;
- использовать инструментальные средства для автоматизации оформления документации;

Технические средства обучения:

персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением

Программное обеспечение:

Embarcadero RAD Studio 2010, 1С: Предприятие 8.3, GIMP, Git, MS Visual Studio 2010

Оборудование лаборатории и рабочих мест лаборатории:

- Пакет прикладных программ «Microsoft Office»: табличный процессор Microsoft Excel, текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer
- Компилятор tasm
- Отладчик машинных кодов AFDP
- MS Visual Studio 2010
- Embarcadero RAD Studio 2010
- 1С: Предприятие 8.3.
- Git.

Реализация рабочей программы ПМ предполагает обязательную учебную и производственную практику.

Оборудование и технологическое оснащение рабочих мест – Полигон учебных баз практики:

персональные компьютеры в локальной сети с доступом к сети Internet – 22 ед. с лицензионным программным и сетевым обеспечением;

локальная сеть с доступом к ресурсам интернет;

сервер

Программное обеспечение:

- Embarcadero RAD Studio 2010,
- 1С: Предприятие 8.3,
- GIMP,
- Git,
- MS Visual Studio 2010.

Задания

Написать программу, которая выполняет вычисления с некоторой точностью. Дана функция, заданная бесконечным рядом. С клавиатуры вводятся x и e (e – точность 0.001–0.00001), нужно вычислить бесконечную сумму с заданной точностью e. Вычисления прекращаются, когда очередное слагаемое по модулю меньше точности. Требуется вывести на экран таблицу значений i и суммы i слагаемых.

Варианты:

1. $f(x) = \sum_{i=0}^{\infty} \frac{(-1)^i x^{2i+1}}{(2i+1)i!}$.

2. $f(x) = \sum_{i=0}^{\infty} \frac{(-1)^i x^{4i+1}}{(2i)!(4i+1)}$.

3. $f(x) = \sum_{i=0}^{\infty} \frac{(-1)^{i+1}(x)^{2i-1}}{(2i-1)(2i+1)!}$.

4. $f(x) = \sum_{k=0}^{\infty} \frac{x^{2k}}{2^k k!}$..

5. $f(x) = \sum_{k=1}^{\infty} \frac{x}{k^3 + k\sqrt{|x|+1}}$..

Порядок выполнения работы:
Составление алгоритма на языке программирования.
Написание программного кода на языке программирования
Отладка программы на ПК
Форма представления результата:
Алгоритм программы
Программный код, составленный на языке программирования.

Тема 01.02.03 Введение в событийно-ориентированное программирование
Практическое занятие № 7, 8, 9
Написание программ с использованием:

1. События
2. Сообщения
3. Обработки событий

Цель работы: Написание программ с использованием обработчика событий.
Выполнив работу, Вы будете:
уметь:

- осуществлять разработку кода программного модуля на современных языках программирования;
- выполнять отладку и тестирование программы на уровне модуля;
- использовать инструментальные средства для автоматизации оформления документации;

Технические средства обучения:

персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением
Программное обеспечение:

Embarcadero RAD Studio 2010, 1С: Предприятие 8.3, GIMP, Git, MS Visual Studio 2010

Оборудование лаборатории и рабочих мест лаборатории:

- Пакет прикладных программ «Microsoft Office»: табличный процессор Microsoft Excel, текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer
- Компилятор tasm
- Отладчик машинных кодов AFDP
- MS Visual Studio 2010
- Embarcadero RAD Studio 2010
- 1С: Предприятие 8.3.
- Git.

Реализация рабочей программы ПМ предполагает обязательную учебную и производственную практику.

Оборудование и технологическое оснащение рабочих мест – Полигон учебных баз практики:

персональные компьютеры в локальной сети с доступом к сети Internet – 22 ед. с лицензионным программным и сетевым обеспечением;

локальная сеть с доступом к ресурсам интернет;

сервер

Программное обеспечение:

- Embarcadero RAD Studio 2010,
- 1С: Предприятие 8.3,
- GIMP,
- Git,
- MS Visual Studio 2010.

Задания

1. Разместить на форме поле ввода, метку и кнопку. Создать обработчик события нажатия на кнопку, который будет копировать текст из поля ввода в метку. Создать обработчик события нажатия кнопки мышки на форме, который будет устанавливать цвет формы и менять текст метки на строку «Начало работы» и очищать поле ввода.
2. Разместить на форме поле ввода и две кнопки с надписями: «блокировать», «разблокировать». Создать обработчики события нажатия на кнопки, которые будут делать активными или неактивными поле ввода. Создать обработчик события нажатия кнопки мышки на форме, который будет устанавливать цвет формы и делать невидимыми все элементы.

3. Реализовать игру минер на поле 3х3 из кнопок. Первоначально все кнопки не содержат надписей. При попытке нажатия на кнопку на ней либо показывается количество мин, либо надпись «мина!» и меняется цвет окна.
4. Разместить на форме четыре кнопки. Написать для каждой обработчик события, который будет менять размеры и местоположение на окне других кнопок.
5. Разместить на форме ряд полей ввода, метку и кнопку. Создать обработчик события нажатия на кнопку, который будет копировать текст из поля ввода в метку. Создать обработчик события нажатия кнопки мышки на форме, который будет устанавливать цвет формы и менять текст метки на строку «Начало работы» и очищать поле ввода.

Порядок выполнения работы:

Составление алгоритма на языке программирования.

Написание программного кода на языке программирования

Отладка программы на ПК

Форма представления результата:

Алгоритм программы

Программный код, составленный на языке программирования

Тема 01.02.4 04 Введение в объектно-ориентированное программирование

Практическое занятие № 10,11,12,13,14,15,16,17

Написание программ с использованием:

1. Инкапсуляция. Функции-элементы
2. Сокрытие данных. Открытые, закрытые, защищенные члены класса.
3. Управление созданием, инициализацией и уничтожением объектов классов. Специальные функции- члены классов: конструктор и деструктор
4. Производные и базовые классы

Цель работы: Написание программ с использованием классов.

Выполнив работу, Вы будете:

уметь:

- осуществлять разработку кода программного модуля на современных языках программирования;
- выполнять отладку и тестирование программы на уровне модуля;
- использовать инструментальные средства для автоматизации оформления документации;

Технические средства обучения:

персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением
Программное обеспечение:

Embarcadero RAD Studio 2010, 1C: Предприятие 8.3, GIMP, Git, MS Visual Studio 2010

Оборудование лаборатории и рабочих мест лаборатории:

- Пакет прикладных программ «Microsoft Office»: табличный процессор Microsoft Excel, текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer
- Компилятор tasm
- Отладчик машинных кодов AFDP
- MS Visual Studio 2010
- Embarcadero RAD Studio 2010
- 1C: Предприятие 8.3.
- Git.

Реализация рабочей программы ПМ предполагает обязательную учебную и производственную практику.

Оборудование и технологическое оснащение рабочих мест – Полигон учебных баз практики:

персональные компьютеры в локальной сети с доступом к сети Internet – 22 ед. с лицензионным программным и сетевым обеспечением;

локальная сеть с доступом к ресурсам интернет;

сервер

Программное обеспечение:

- Embarcadero RAD Studio 2010,
- 1C: Предприятие 8.3,
- GIMP,
- Git,
- MS Visual Studio 2010.

Задания

Создать класс A с полями a и b и свойством c. Свойство – значение выражения над полями a и b (выражение и типы полей – см. вариант в таблице 1). Поля инициализировать при объявлении класса. Конструктор оставить по умолчанию. Проследить, чтобы поля a и b напрямую в других классах были недоступны. Создать класс Programm с одним методом – точкой входа. В теле метода создать объект класса A, вывести на экран значение свойства c.

Вариант	Тип полей a b,	Выражения
1	float	/,-
2	float	/=, +
3	int	*,-
4	int	*=,+

5 int (постфиксный)++,*=,-

Порядок выполнения работы:

Составление алгоритма на языке программирования.

Написание программного кода на языке программирования

Отладка программы на ПК

Форма представления результата:

Алгоритм программы

Программный код, составленный на языке программирования

Тема 01.02.07 Расширения управляемого C++ (Managed C++)

Практическое занятие № 18,19

Написание программ с использованием:

1. Управляемые массивы.
2. Управляемые двумерные массивы

Цель работы: Написание программ с использованием управляемых массивов и двумерных массивов.

Выполнив работу, Вы будете:

уметь:

- осуществлять разработку кода программного модуля на современных языках программирования;
- выполнять отладку и тестирование программы на уровне модуля;
- использовать инструментальные средства для автоматизации оформления документации;

Технические средства обучения:

персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением

Программное обеспечение:

Embarcadero RAD Studio 2010, 1С: Предприятие 8.3, GIMP, Git, MS Visual Studio 2010

Оборудование лаборатории и рабочих мест лаборатории:

- Пакет прикладных программ «Microsoft Office»: табличный процессор Microsoft Excel, текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer
- Компилятор tasm
- Отладчик машинных кодов AFDP
- MS Visual Studio 2010
- Embarcadero RAD Studio 2010
- 1С: Предприятие 8.3.
- Git.

Реализация рабочей программы ПМ предполагает обязательную учебную и производственную практику.

Оборудование и технологическое оснащение рабочих мест – Полигон учебных баз практики:
персональные компьютеры в локальной сети с доступом к сети Internet – 22 ед. с лицензионным программным и сетевым обеспечением;
локальная сеть с доступом к ресурсам интернет;

сервер

Программное обеспечение:

- Embarcadero RAD Studio 2010,
- 1С: Предприятие 8.3,
- GIMP,
- Git,
- MS Visual Studio 2010.

Задания

Даны натуральные *n, m*, целые числа *a*1, …, *an* и *b*1, …, *bm*. Внутри каждой из данных последовательностей нет повторяющихся членов. Построить пересечение данных последовательностей.

2. Даны натуральные *n, m*, целые числа *a*1, …, *an* и *b*1, …, *bm*. Внутри каждой из данных последовательностей нет повторяющихся членов. Получить все члены последовательности *a*1, …, *an*, которые не входят в последовательность *b*1, …, *bm*.

3. Даны массив $x[1] \le x[2] \le ... \le x[n]$ целых чисел и число *y*.Выяснить, содержится ли *y* в этом массиве, то есть существует ли $x[i] = y$, *i* из 1..*n*. Число действий в программе порядка *log n*.

4. Даны массив $x[1...n]$ целых чисел и число *y*. Переставить числа в массиве таким образом, чтобы слева от некоторой границы стояли числа, меньшие или равные *y*, а справа – большие или равные *y*.

5. Даны натуральное *n*, целые числа *a*1, …, *an*. Если в данной последовательности ни одно четное число не расположено после нечетного, то вывести на экран все отрицательные члены последовательности в обратном порядке, иначе – все положительны

Создайте приложение, при выполнении которого происходит:
а) ввод целых чисел в двумерный массив, состоящий из 5 строк по 3 элемента в каждой, с нижними границами индексов, равными 1,причем в объявлении массива должны быть использованы идентификаторы именованных констант для указания значений границ индексов массива;
б) ввод целых чисел в одномерный массив (число элементов которого должно быть равно числу столбцов двумерного массива и в объявлении этого одномерного массива должны быть использованы идентификаторы именованных констант для указания значений границ его индексов);
в) сортировка элементов в столбцах двумерного массива или невыполнение сортировки в зависимости от значений соответствующих элементов

одномерного массива и затем вывод результатов (если, например, k-й элемент одномерного массива отрицательный, то выполняется сортировка по убыванию значений элементов k-го столбца двумерного массива, если этот элемент равен нулю, то сортировка не выполняется, если положительный, то выполняется сортировка по возрастанию).

Порядок выполнения работы:

Составление алгоритма на языке программирования.

Написание программного кода на языке программирования

Отладка программы на ПК

Форма представления результата:

Алгоритм программы

Программный код, составленный на языке программирования

Тема 01.02.08 Работа со строками в Windows

Практическое занятие № 20,21,22,23

Написание программ с использованием:

1. Перекодирование однобайтовых символов в Unicode и обратно с учетом кодовой страницы.
2. Инициализация строк при помощи строковых констант.
3. Методы класса String. Составное форматирование. Разбор строк.

Цель работы: Написание программ с использованием различных приложений.

Выполнив работу, Вы будете:

уметь:

- осуществлять разработку кода программного модуля на современных языках программирования;
- выполнять отладку и тестирование программы на уровне модуля;
- использовать инструментальные средства для автоматизации оформления документации;

Технические средства обучения:

персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением

Программное обеспечение:

Embarcadero RAD Studio 2010, 1С: Предприятие 8.3, GIMP, Git, MS Visual Studio 2010

Оборудование лаборатории и рабочих мест лаборатории:

- Пакет прикладных программ «Microsoft Office»: табличный процессор Microsoft Excel, текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer
- Компилятор tasm
- Отладчик машинных кодов AFDP

- MS Visual Studio 2010
- Embarcadero RAD Studio 2010
- 1С: Предприятие 8.3.
- Git.

Реализация рабочей программы ПМ предполагает обязательную учебную и производственную практику.

Оборудование и технологическое оснащение рабочих мест – Полигон учебных баз практики:

персональные компьютеры в локальной сети с доступом к сети Internet – 22 ед. с лицензионным программным и сетевым обеспечением;

локальная сеть с доступом к ресурсам интернет;

сервер

Программное обеспечение:

- Embarcadero RAD Studio 2010,
- 1С: Предприятие 8.3,
- GIMP,
- Git,
- MS Visual Studio 2010.

Задания

1. Найти число тех групп букв, которые заканчиваются той же буквой, что и первая группа букв.
2. Если в данном тексте есть группа знаков, начинающаяся знаком «*», то в следующей по порядку группе знаков (если она существует) заменить каждый из знаков нулем.
3. Вывести на экран те слова, которые отличны от последнего слова текста и все буквы этого слова различны.
4. Если в тексте нет символа «+», то оставить текст без изменения, иначе каждую из цифр, предшествующую первому вхождению знака «+», заменить на «*».
5. Вывести на экран текст, составленный из последних букв всех групп букв данного файла.

Порядок выполнения работы:

Составление алгоритма на языке программирования.

Написание программного кода на языке программирования

Отладка программы на ПК

Форма представления результата:

Алгоритм программы

Программный код, составленный на языке программирования

Тема 01.02.09 Принципы разработки Windows приложений в .NET (Windows Forms Application)

Практическое занятие № 24,25,26,27

Написание программ с использованием:

1. Формы. Компонентная модель.
2. Конструирование приложения.
3. Генерация кода
4. Режимы дизайна и кода

Цель работы: Написание программ с использованием различных приложений.

Выполнив работу, Вы будете:

уметь:

- осуществлять разработку кода программного модуля на современных языках программирования;
- выполнять отладку и тестирование программы на уровне модуля;
- использовать инструментальные средства для автоматизации оформления документации;

Технические средства обучения:

персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением

Программное обеспечение:

Embarcadero RAD Studio 2010, 1С: Предприятие 8.3, GIMP, Git, MS Visual Studio 2010

Оборудование лаборатории и рабочих мест лаборатории:

- Пакет прикладных программ «Microsoft Office»: табличный процессор Microsoft Excel, текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer
- Компилятор tasm
- Отладчик машинных кодов AFDP
- MS Visual Studio 2010
- Embarcadero RAD Studio 2010
- 1С: Предприятие 8.3.
- Git.

Реализация рабочей программы ПМ предполагает обязательную учебную и производственную практику.

Оборудование и технологическое оснащение рабочих мест – Полигон учебных баз практики:

персональные компьютеры в локальной сети с доступом к сети Internet – 22 ед. с лицензионным программным и сетевым обеспечением;

локальная сеть с доступом к ресурсам интернет;

сервер

Программное обеспечение:

- Embarcadero RAD Studio 2010,

– 1С: Предприятие 8.3,
– GIMP,
– Git,
– MS Visual Studio 2010.

Задания

Программа представляет собой автоматизированную систему учета банковских сведений.

На каждого клиента банка хранятся следующие сведения:

- Ф.И.О.;
- Возраст;
- Место работы;
- Номера счетов.

На каждом счете хранится информация о текущем балансе и история прихода, расхода. Для каждого клиента может быть создано неограниченное количество счетов. С каждым счетом можно производить следующие действия: открытие, закрытие, вклад денег, снятие денег, просмотр баланса, просмотр истории. Вся информация должна хранится в массивах. Рекомендуется объекты клиента и счета реализовать в виде классов. Баланс счета организовать в виде свойства только для чтения.

Создать интерфейс к программе Ввод сведений о новых клиентах и счетах реализовать через дополнительные формы.

Порядок выполнения работы:

Составление алгоритма на языке программирования.

Написание программного кода на языке программирования

Отладка программы на ПК

Форма представления результата:

Алгоритм программы

Программный код, составленный на языке программирования

Тема 01.02.10 Окна инструментов среды разработки Visual Studio

Практическое занятие № 28,29

Написание программ с использование:

1. Интерфейс окна Properties. Обработка событий
2. Обработка событий. Окно Server Explorer

Цель работы: Написание программ с использованием обработки событий.

Выполнив работу, Вы будете:

уметь:

- осуществлять разработку кода программного модуля на современных языках программирования;

- выполнять отладку и тестирование программы на уровне модуля;
- использовать инструментальные средства для автоматизации оформления документации;

Технические средства обучения:
персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением
Программное обеспечение:
Embarcadero RAD Studio 2010, 1С: Предприятие 8.3, GIMP, Git, MS Visual Studio 2010
Оборудование лаборатории и рабочих мест лаборатории:

- Пакет прикладных программ «Microsoft Office»: табличный процессор Microsoft Excel, текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer
- Компилятор tasm
- Отладчик машинных кодов AFDP
- MS Visual Studio 2010
- Embarcadero RAD Studio 2010
- 1С: Предприятие 8.3.
- Git.

Реализация рабочей программы ПМ предполагает обязательную учебную и производственную практику.
Оборудование и технологическое оснащение рабочих мест – Полигон учебных баз практики:
персональные компьютеры в локальной сети с доступом к сети Internet – 22 ед. с лицензионным программным и сетевым обеспечением;
локальная сеть с доступом к ресурсам интернет;
сервер
Программное обеспечение:

- Embarcadero RAD Studio 2010,
- 1С: Предприятие 8.3,
- GIMP,
- Git,
- MS Visual Studio 2010.

Задания
Построить иерархию классов в соответствии с вариантом задания:
1) Студент, преподаватель, персона, заведующий кафедрой
2) Служащий, персона, рабочий, инженер
3) Рабочий, кадры, инженер, администрация
4) Деталь, механизм, изделие, узел
5) Организация, страховая компания, нефтегазовая компания, завод

Расширить иерархию классов с использованием виртуального класса в качестве основы иерархии.
Реализовать для иерархии из лабораторной работы механизм интерфейсов, при этом один из классов должен реализовывать как минимум 2 интерфейса. Использовать для проверки всех методов данного класса многоадресный делегат.

Порядок выполнения работы:

Составление алгоритма на языке программирования.
Написание программного кода на языке программирования
Отладка программы на ПК

Форма представления результата:

Алгоритм программы
Программный код, составленный на языке программирования

Тема 01.02.11 Перемещение и изменение размеров окон инструментов
Практическое занятие № 30,31

Написание программ с использованием стандартных элементов управления:

1. Закрепление инструмента в Visual Studio
2. Скрытие инструмента в Visual Studio

Цель работы: Написание программ для работы в Visual Studio

Выполнив работу, Вы будете:

уметь:

- осуществлять разработку кода программного модуля на современных языках программирования;
- выполнять отладку и тестирование программы на уровне модуля;
- использовать инструментальные средства для автоматизации оформления документации;

Технические средства обучения:
персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением
Программное обеспечение:
Embarcadero RAD Studio 2010, 1С: Предприятие 8.3, GIMP, Git, MS Visual Studio 2010
Оборудование лаборатории и рабочих мест лаборатории:

- Пакет прикладных программ «Microsoft Office»: табличный процессор Microsoft Excel, текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer
- Компилятор tasm
- Отладчик машинных кодов AFDP
- MS Visual Studio 2010

– Embarcadero RAD Studio 2010
– 1С: Предприятие 8.3.
– Git.

Реализация рабочей программы ПМ предполагает обязательную учебную и производственную практику.

Оборудование и технологическое оснащение рабочих мест – Полигон учебных баз практики:

персональные компьютеры в локальной сети с доступом к сети Internet – 22 ед. с лицензионным программным и сетевым обеспечением;

локальная сеть с доступом к ресурсам интернет;

сервер

Программное обеспечение:

– Embarcadero RAD Studio 2010,
– 1С: Предприятие 8.3,
– GIMP,
– Git,
– MS Visual Studio 2010.

Задания

При выполнении работы необходимо определить базовый класс и производные от него классы. Предусмотреть передачу аргументов конструкторам базового класса; использование виртуальных и перегруженных функций; обработку исключительных ситуаций.

В задании требуется создать базовый класс (как вариант абстрактный базовый класс) и определить общие методы show ( ), get ( ), set ( ) и другие, специфические для данного класса. Создать производные классы, в которые добавить свойства и методы. Часть методов переопределить. Создать массив объектов базового класса и заполнить объектами производных классов. Объекты производных классов идентифицировать конструктором по имени или идентификационному номеру.

Вызвать метод show ( ) базового класса и просмотреть массив объектов.

Использовать объекты для моделирования реальных ситуаций.

1. Создать базовый класс «Транспортное средство» и производные классы «Автомобиль», «Велосипед», «Повозка». Подсчитать время и стоимость перевозки пассажиров и грузов каждым транспортным средством.
2. Создать базовый класс «Грузоперевозчик» и производные классы «Самолет», «Поезд», «Автомобиль». Определить время и стоимость перевозки для указанных городов и расстояний.
3. Создать аналогичный базовый класс «Пассажироперевозчик» и производные классы «Самолет», «Поезд», «Автомобиль». Определить время и стоимость передвижения.

4. Создать базовый класс «Учащийся» и производные классы «Школьник» и «Студент». Создать массив объектов базового класса и заполнить этот массив объектами. Показать отдельно студентов и школьников.
5. Создать базовый класс «Музыкальный инструмент» и производные классы «Ударный», «Струнный», «Духовой». Создать массив объектов «Оркестр». Выдать состав оркестра, переопределив метод.

Порядок выполнения работы:

Составление алгоритма на языке программирования.
Написание программного кода на языке программирования
Отладка программы на ПК

Форма представления результата:

Алгоритм программы
Программный код, составленный на языке программирования

Тема 01.02.12 Контроль версий GIT

Практическое занятие № 32,33,34,№5,36,37,38,39

Написание программ с использованием контроля версий GIT

1. Создание репозитария. Проверка состояния репозитария
2. Внесение изменений. Индексация изменений
3. Индексация и коммит. Коммит изменений
4. Коммит второго изменения
5. Контроль отображения записей

Цель работы: Написание программ с использованием контроля версий GIT.

Выполнив работу, Вы будете:

уметь:

- осуществлять разработку кода программного модуля на современных языках программирования;
- выполнять отладку и тестирование программы на уровне модуля;
- использовать инструментальные средства для автоматизации оформления документации;

Технические средства обучения:
персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением
Программное обеспечение:

Embarcadero RAD Studio 2010, 1С: Предприятие 8.3, GIMP, Git, MS Visual Studio 2010

Оборудование лаборатории и рабочих мест лаборатории:

– Пакет прикладных программ «Microsoft Office»: табличный процессор Microsoft Excel, текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer

– Компилятор tasm
– Отладчик машинных кодов AFDP
– MS Visual Studio 2010
– Embarcadero RAD Studio 2010
– 1С: Предприятие 8.3.
– Git.

Реализация рабочей программы ПМ предполагает обязательную учебную и производственную практику.

Оборудование и технологическое оснащение рабочих мест – Полигон учебных баз практики:

персональные компьютеры в локальной сети с доступом к сети Internet – 22 ед. с лицензионным программным и сетевым обеспечением;

локальная сеть с доступом к ресурсам интернет;

сервер

Программное обеспечение:

– Embarcadero RAD Studio 2010,
– 1С: Предприятие 8.3,
– GIMP,
– Git,
– MS Visual Studio 2010.

Задания

Лабораторная работа №1 (GIT)

01 Установка имени и электронной почты

Если вы никогда ранее не использовали **git,** для начала необходимо осуществить установку. Выполните следующие команды, чтобы **git** узнал ваше имя и электронную почту. Если **git** уже установлен, можете переходить к разделу окончания строк.

Выполнить:

```
git config --global user.name "Your Name"
git config --global user.email
your email@whatever.com
```

02 Параметры установки окончаний строк

Также, для пользователей Unix/Mac:

Выполнить:

```
git config --global core.autocrlf input
git config --global core.safecrlf true
```

Для пользователей Windows:

Выполнить:

```
git config --global core.autocrlf true
git config --global core.safecrlf true
```

03 Установка отображения unicode

По умолчанию, **git** будет печатать не-ASCII символов в именах файлов в виде восьмеричных последовательностей `\nnn`. Что бы избежать нечитаемых строк, установите соответствующий флаг.

```
git config --global core.quotepath off
```

Лабораторная работа №2 (GIT)

01 Скачайте учебные материалы

Скачайте учебные материалы здесь:http://githowto.com/git_tutorial.zip

02 Распакуйте учебные материалы

Пакет учебных материалов должен иметь главную папку «git_tutorial» с двумя подпапками:

- work — пустой рабочий каталог. Здесь будут лежать ваши репозитории.

files — заранее упакованные файлы для того, чтобы вы могли продолжить работать с учебными материалами на любом этапе. Если вы застрянете, просто скопируйте нужный урок в свою рабочую папку.

Создание проекта

01 Создайте страницу «Hello, World»

Начните работу в пустом рабочем каталоге (например Work, если вы скачали архив с предыдущего шага) с создания пустого каталога с именем «hello», затем войдите в него и создайте там файл с именем **`hello.html`** с таким содержанием.

Выполните:

```
mkdir hello
cd hello
touch hello.html
```

Файл: ***hello.html***

```
Hello, World
```

02 Создайте репозиторий

Теперь у вас есть каталог с одним файлом. Чтобы создать git репозиторий из этого каталога, выполните команду **git init.**

Выполните:

```
git init
```

Результат:

```
$ git init
Initialized empty Git repository in
/Users/alex/Documents/Presentations/githowto/auto/he
llo/.git/
```

03 Добавьте страницу в репозиторий

Теперь добавим в репозиторий страницу «Hello, World».

Выполните:

```
git add hello.html
```

```
git commit -m "First Commit"
```
Вы увидите …

Результат:

```
$ git add hello.html
$ git commit -m "First Commit"
[master (root-commit) 911e8c9] First Commit
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 hello.html
```

4. Проверка состояния

01 Проверьте состояние репозитория

Используйте команду **`git status`,** чтобы проверить текущее состояние репозитория.

Выполните:

```
git status
```
Вы увидите

Результат:

```
$ git status
# On branch master
nothing to commit (working directory clean)
```
Команда проверки состояния сообщит, что коммитить нечего. Это означает, что в репозитории хранится текущее состояние рабочего каталога, и нет никаких изменений, ожидающих записи.

Мы будем использовать команду **`git status`**, чтобы продолжать отслеживать состояние репозитория и рабочего каталога.

Лабораторная работа №3(GIT)

5. Внесение изменений

01 Измените страницу «Hello, World»

Добавим кое-какие HTML-теги к нашему приветствию. Измените содержимое файла на:

Файл: *hello.html*

```
<h1>Hello, World!</h1>
```

02 Проверьте состояние

Теперь проверьте состояние рабочего каталога.

Выполните:

```
git status
```
Вы увидите …

Результат:

```
$ git status
# On branch master
# Changes not staged for commit:
```

```
#   (use "git add <file>..." to update what will be
committed)
#   (use "git checkout -- <file>..." to discard
changes in working directory)
#
#   modified:   hello.html
#
no changes added to commit (use "git add" and/or
"git commit -a")
```

Первое, что нужно заметить, это то, что **git** знает, что файл **hello.html** был изменен, но при этом эти изменения еще не зафиксированы в репозитории.

Также обратите внимание на то, что сообщение о состоянии дает вам подсказку о том, что нужно делать дальше. Если вы хотите добавить эти изменения в репозиторий, используйте команду **git add**. В противном случае используйте команду **git checkout** для отмены изменений.

6. Индексация изменений

01 Добавьте изменения

Теперь дайте команду **git** проиндексировать изменения. Проверьте состояние

Выполните:

```
git add hello.html
git status
```

Вы увидите…

Результат:

```
$ git add hello.html
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.html
#
```

Изменения файла **hello.html** были проиндексированы. Это означает, что **git** теперь знает об изменении, но изменение пока не *перманентно* (читай, *навсегда*) записано в репозиторий. Следующий коммит будет включать в себя проиндексированные изменения.

Если вы решили, что *не* хотите коммитить изменения, команда состояния напомнит вам о том, что с помощью команды **git reset** можно снять индексацию этих изменений.

7. Индексация и коммит

Отдельный шаг индексации в git позволяет вам продолжать вносить изменения в рабочий каталог, а затем, в момент, когда вы захотите взаимодействовать с версионным контролем, git позволит записать изменения в малых коммитах, которые фиксируют то, что вы сделали.
Предположим, что вы отредактировали три файла **(a.html, b.html, и c.html).** Теперь вы хотите закоммитить все изменения, при этом чтобы изменения в `a.html` и `b.html` были одним коммитом, в то время как изменения в `c.html` логически не связаны с первыми двумя файлами и должны идти отдельным коммитом.
В теории, вы можете сделать следующее:

```
git add a.html
git add b.html
git commit -m "Changes for a and b"
git add c.html
git commit -m "Unrelated change to c"
```

Разделяя индексацию и коммит, вы имеете возможность с легкостью настроить, что идет в какой коммит.

Лабораторная работа №4 (GIT)

8. Коммит изменений

01 Закоммитьте изменения

Сделаем коммит того, что мы проиндексировали, в репозиторий.
Когда ранее использовали **`git commit`** для коммита первоначальной версии файла **`hello.html`** в репозиторий, вы включили метку **`-m`**, которая делает комментарий в командной строке. Команда **commit** позволит интерактивно редактировать комментарии для коммита.
Если опустить метку **`-m`** из командной строки, **git** перенесет вас в редактор по вашему выбору. Редактор выбирается из следующего списка (в порядке приоритета):

- переменная среды GIT_EDITOR
- параметр конфигурации core.editor
- переменная среды VISUAL
- переменная среды EDITOR

Переменная EDITOR установлена в **`emacsclient`** (доступен для Linux и Mac).
Сделайте коммит сейчас и проверьте состояние.
Выполните:

```
git commit
```

Вы увидите в вашем редакторе:
Результат:

```
|
```

```
# Please enter the commit message for your changes.
Lines starting
# with '#' will be ignored, and an empty message
aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.html
#
```

В первой строке введите комментарий: «Added h1 tag». Сохраните файл и выйдите из редактора (для этого в редакторе по-умолчанию (Vim) вам нужно нажать клавишу ESC, ввести `:wq` и нажать Enter). Вы увидите…

Результат:

```
git commit
Waiting for Emacs...
[master 569aa96] Added h1 tag
 1 files changed, 1 insertions(+), 1 deletions(-)
```

Строка «Waiting for Emacs…» получена из программы **emacsclient**, которая посылает файл в запущенную программу **emacs** и ждет его закрытия. Остальные выходные данные – стандартные коммит - сообщения.

02 Проверьте состояние

В конце давайте еще раз проверим состояние.

Выполните:

```
git status
```

Вы увидите…

Результат:

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

Рабочий каталог чистый, можете продолжить работу.

Лабораторная работа №5 (GIT)

9. Изменения, а не файлы

Большинство систем версионного контроля работают с файлами. Вы добавляете файл в версионный контроль, а система отслеживает изменения файла с этого момента.

01 Первое изменение: Добавьте стандартные теги страницы

Измените страницу «Hello, World», чтобы она содержала стандартные теги **<html>** и **<body>.**

Файл: *hello.html*

```
<html>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

02 Добавьте это изменение

Теперь добавьте это изменение в индекс git.

Выполните:

```
git add hello.html
```

03 Второе изменение: Добавьте заголовки HTML

Теперь добавьте заголовки HTML (секцию **<head>**) к странице «Hello, World».

Файл: ***hello.html***

```
<html>
  <head>
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

04 Проверьте текущий статус

Выполните:

```
git status
```

Вы увидите…

Результат:

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.html
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be
committed)
#   (use "git checkout -- <file>..." to discard
changes in working directory)
#
#   modified:   hello.html
#
```

Обратите внимание на то, что `hello.html` указан дважды в состоянии. Первое изменение (добавление стандартных тегов) проиндексировано и готово к коммиту. Второе изменение (добавление заголовков HTML) яв-

ляется непроиндексированным. Если бы вы делали коммит сейчас, заголовки не были бы сохранены в репозиторий.
Проверим.

05 Коммит

Произведите коммит проиндексированного изменения (значение по умолчанию), а затем еще раз проверьте состояние.

Выполните:

```
git commit -m "Added standard HTML page tags"
git status
```

Вы увидите…

Результат:

```
$ git commit -m "Added standard HTML page tags"
[master 8c32287] Added standard HTML page tags
 1 files changed, 3 insertions(+), 1 deletions(-)
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be
committed)
#   (use "git checkout -- <file>..." to discard
changes in working directory)
#
#   modified:   hello.html
#
no changes added to commit (use "git add" and/or
"git commit -a")
```

Состояние команды говорит о том, что `hello.html` имеет незафиксированные изменения, но уже не в буферной зоне.

06 Добавьте второе изменение

Теперь добавьте второе изменение в индекс, а затем проверьте состояние с помощью команды **`git status`**.

Выполните:

```
git add .
git status
```

Примечание: В качестве файла для добавления, мы использовали текущий каталог («.»). Это самый краткий и удобный путь для добавления всех изменений в файлы текущего каталога и его подкаталоги. Но поскольку он добавляет все, *не лишним* будет проверить состояние перед запуском `add`, просто чтобы убедиться, что вы не добавили какой-то файл, который добавлять было не нужно.

Вы увидите…

Результат:

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.html
#
```

Второе изменение было проиндексировано и готово к коммиту.

07 Сделайте коммит второго изменения

Выполните:

```
git commit -m "Added HTML header"
```

Порядок выполнения работы:

Составление алгоритма на языке программирования.

Написание программного кода на языке программирования.

Отладка программы на ПК.

Форма представления результата:

Алгоритм программы.

Программный код, составленный на языке программирования.

Тема 01.02.13 1С:Предприятие. Создание информационной базы. Подсистемы. Справочники. Документы.

Практическое занятие №40,41,42,43,44

Написание программ с использованием программы 1С:Предприятие

1. Создание новой информационной базы. Знакомство с конфигуратором. Дерево объектов
2. Добавление подсистемы. Порядок разделов.
3. Создание справочника. Справочник с табличной частью.
4. Иерархический справочник
5. Создание документов

Цель работы: Написание программ с использованием программы 1С:Предприятие.

Выполнив работу, Вы будете:

уметь:

- осуществлять разработку кода программного модуля на современных языках программирования;
- выполнять отладку и тестирование программы на уровне модуля;
- использовать инструментальные средства для автоматизации оформления документации;

Технические средства обучения:

персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением

Программное обеспечение:

Embarcadero RAD Studio 2010, 1С: Предприятие 8.3, GIMP, Git, MS Visual Studio 2010

Оборудование лаборатории и рабочих мест лаборатории:

- Пакет прикладных программ «Microsoft Office»: табличный процессор Microsoft Excel, текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer
- Компилятор tasm
- Отладчик машинных кодов AFDP
- MS Visual Studio 2010
- Embarcadero RAD Studio 2010
- 1С: Предприятие 8.3.
- Git.

Реализация рабочей программы ПМ предполагает обязательную учебную и производственную практику.

Оборудование и технологическое оснащение рабочих мест – Полигон учебных баз практики:

персональные компьютеры в локальной сети с доступом к сети Internet – 22 ед. с лицензионным программным и сетевым обеспечением;

локальная сеть с доступом к ресурсам интернет;

сервер

Программное обеспечение:

- Embarcadero RAD Studio 2010,
- 1С: Предприятие 8.3,
- GIMP,
- Git,
- MS Visual Studio 2010.

Задания

Выполнение задания состоит из следующих этапов.

1. Сформировать новую конфигурацию в режиме УП.
2. Создать в конфигурации подсистемы: «Лабораторная работа», «Администрирование». Расположить их на панели разделов в указанной последовательности.
3. Создать две роли: Администратор, наделенный всеми правами, в т. числе и административными, и пользователь без административных прав.
4. Создать двух пользователей, один из которых должен играть роль администратора, а второй – пользователя.
5. Создать константу, указанную в задании, включив ее в подсистему «Администрирование». Дополнительно создать форму констант, включив ее в подсистему «Администрирование». В пользовательском режиме за-

дать значение данной константы. В дальнейшем при запуске приложения выводить значение данной константы в окно сообщений. Если для разработки типа константы необходимо разработать дополнительные объекты, включить их в подсистему «Лабораторная работа».

6. Разработать справочники, указанные в задании. Включить их в подсистему «Лабораторная работа». Основной (первый в каждом задании) справочник должен быть многоуровневым. Количество уровней - произвольно. Название групп в справочнике - произвольно.

Решение задачи состоит из следующих этапов:

- Описать структуры справочников средствами конфигуратора;
- Разработать формы диалога ввода данных в справочники (как для ввода групп, так и для ввода элементов). Группы справочников должны состоять только из кодов и наименований;
- В режиме 1С:Предприятие ввести несколько групп, содержащий не менее 2-3 элементов.
- Дополнительно в случае необходимости разработать объекты метаданных, необходимые для работы заданного справочника в Конфигурации.
- Обеспечить проверку заполнения всех реквизитов шапки основного справочника. Разрешать запись элемента справочника только в случае заполненности всех реквизитов шапки.
- Поместить команду создания нового элемента основного справочника в задании на рабочий стол приложения.
- В форму списка основного справочника вставить кнопку с названием «Справка». При нажатии на эту кнопку программа должна вывести в окно сообщений справку. Содержание справки – см. варианты заданий. Использовать команду Сообщить(…)
- Для разработанного основного справочника сформировать подчиненный справочник. Структура справочника – см. варианты.
- Проверку заполненности трех обязательных (произвольных) реквизитов основного справочника перед записью элемента справочника выполнить программным путем, поместив процедуру проверки в модуль менеджера.

7. Создать обработку, которая должна выводить список всех справочников, разработанных в конфигурации. Для вывода информации использовать команду Сообщить(…)

Вариант 1

1. Константа «Главный бухгалтер», тип СправочникСсылка.Сотрудники

2. Справочник основных средств (ОС):

- Инвентарный номер ОС;

– Наименование ОС;
– Основное материально ответственное лицо (элемент справочника сотрудников);
– Сумма первоначального износа;
– Дата ввода в эксплуатацию;
– Год выпуска.
– Срок эксплуатации (в мес.)

Табличная часть элементов справочника содержит список подразделений, которым принадлежало ОС в течение времени эксплуатации и имеет следующую структуру:
– Дата появления в подразделении
– Подразделение (элемент справочника подразделений).

3. Справочник материально-ответственных лиц (подчиненный):
– материально ответственное лицо (элемент справочника сотрудников);
– Домашний адрес

Замечание 1. При вводе нового элемента справочника обеспечить автоматическое заполнение реквизита "Материально ответственное лицо" предопределенным значением из списка сотрудников (по умолчанию).

Замечание 2. Сформировать предопределенную группу в справочнике «ОС на консервации» ***Получить справку: вывести наименования ОС, год даты ввода в эксплуатацию которых позднее 1980 года.***

Вариант 2

1. Константа «Директор», тип СправочникСсылка.Сотрудники

2. Справочник товаров:
– Код товара;
– Наименование товара;
– Основная единица измерения (элемент справочника единиц измерения);
– Тип товара (перечисление со значениями: весовой, невесовой).

Табличная часть элементов справочника содержит список возможных единиц измерений каждого товара и имеет следующую структуру:
– единица измерения (элемент справочника единиц измерения;
– Коэффициент пересчета относительно основной единицы измерения;
– Признак основной единицы измерения.

3. Справочник цен товаров (подчиненный):

- Тип цены (перечисление со значениями: ***закупочная, розничная, оптовая***) ;
- значение цены.

Замечание. При вводе нового элемента справочника обеспечить автоматическое заполнение реквизита "Основная единица измерения" предопределенным значением из списка единиц измерения (по умолчанию).

Замечание 2. Сформировать предопределенную группу в справочнике «Неликвидные товары»

Получить справку: вывести наименования всех товаров, у которых розничная ценаменьше, 1.5*ЗакупочнойЦены.

Вариант 3

1. Константа «Главный бухгалтер», тип СправочникСсылка.Сотрудники

2 Справочник сотрудников:

- Табельный номер сотрудника;
- ФИО сотрудника;
- Тип сотрудника (выбирается из списка – перечисления со значениями: штатный, внутренний совместитель, внешний совместитель);
- Дата поступления на работу;
- Оклад;
- Признак членства в профсоюзе.

Табличная часть элементов справочника содержит список детей сотрудника и имеет следующую структуру:

- Пол (выбирается из списка, заданного перечислением);
- ФИО ребенка.
- Дата рождения

3 Справочник стандартных вычетов сотрудника (подчиненный):

- Тип вычета (выбирается из списка, задаваемого перечислением со значениями: Собственный, на ребенка, Герой России);
- Сумма вычета.
- Дата начала действия
- Дата окончания действия

Замечание. При вводе нового элемента справочника обеспечить автоматическое заполнение реквизита "Тип сотрудника" предопределенным значением - "ИТР" (по умолчанию).

Замечание 2. Сформировать предопределенную группу в справочнике «Уволенные»

Получить справку: вывести наименования всех сотрудников – не членов профсоюза

Вариант 4

1. Константа «Главный контрагент», тип СправочникСсылка.Контрагенты

2. Справочник контрагентов:

- Код контрагента;
- Наименование контрагента;
- Адрес контрагента;
- Основной расчетный счет(элемент справочника расчетных счетов контрагента);
- Банк контрагента (элемент справочника банков)
- Телефон ;
- Размер кредита.

Табличная часть элементов справочника содержит список договоров контрагента и имеет следующую структуру:

- Код договора;
- Наименование договора.
- Дата заключения договора
- условия договора (текст)

3. Справочник расчетных счетов контрагента(подчиненный)

- код
- наименование
- банк (элемент справочника банков)

Замечание. При вводе нового элемента справочника обеспечить автоматическое заполнение реквизита "Размер кредита" предопределенным значением - "5 000 руб" (по умолчанию).

Замечание 2. Сформировать предопределенную группу в справочнике «Зарубежные контрагенты»

Получить справку: вывести наименования всех контрагентов с нулевым значением кредита.

Вариант 5

1. Константа «Основное транспортное средство директора», тип СправочникСсылка.ТранспортныеСредстваПредприятия

2. Справочник ТранспортныеСредстваПредприятия

- Тип транспортного средства (выбирается из списка - перечисления);
- Наименование транспортного средства;
- Объем двигателя;
- Мощность двигателя;
- Дата выпуска.

– Водитель (элемент справочника Сотрудников)

Табличная часть элементов справочника содержит график прохождения технического осмотра (ТО) транспортного средства и имеет следующую структуру:

– Номер по порядку;
– Дата прохождения ТО;
– Примечание (текст).

3. Справочник товарно-транспортных накладных (подчиненный)

– номер накладной
– Дата выписки накладной
– Время пребывания в пути
– Стоимость израсходованного горючего

Замечание. При вводе нового элемента справочника обеспечить автоматическое заполнение реквизита "Тип транспортного средства" предопределенным значением – из соответствующего списка типов (по умолчанию).

Замечание 2. Сформировать предопределенную группу в справочнике «транспортные средства по лизингу»

Получить справку: вывести наименования всех транспортных средств с объемом двигателя меньше 40 л.с.

1. Разработать новую подсистему. Все новые объекты данного задания поместить в данную подсистему.
2. Для обеспечения работы документа сформировать дополнительные объекты данных (константы, справочники, перечисления).
3. В форме документа необходимо организовать две закладки: «Шапка» (содержит реквизиты шапки) и «Таблица» (содержит табличную часть документа.).
4. Ввести в конфигурацию общий реквизит для всех документов конфигурации. Название и тип- см. варианты заданий.
5. В форме документа должно быть поле, показывающее итоговую сумму по какому-либо числовому полю табличной части документа.
6. Сформировать печатную форму документа. Вызов – по кнопке «Печать», расположенной в форме документа. Использовать конструктор запроса с обработкой результата.
7. Сформировать обработку «Работа с документами». Задачи, возложенные на данную обработку – см. варианты заданий. Поместить в подсистему «Лабораторная работа».
8. В пользовательском режиме сформировать и записать несколько документов.

9. С помощью механизма функциональных опций обеспечить при внедрении приложения на предприятии возможность его настройки на особенности ведения учета. Настраиваемая особенность - см. варианты заданий.

Вариант 1
1. Общий реквизит – Автор, тип «СправорчникСсылка.Пользователи». Означивается автоматически при открытии нового документа именем пользователя, с которым зашли в задачу.
2. Документ «Поступление товаров от поставщиков»
Поставщик_____
Склад_____

Товар Цена Количество Единица измерения Стоимость

- При вводе нового документа поле «Поставщик» означивается по умолчанию выбранным значением из справочника поставщиков.
- При выборе товара автоматически означиваются поля «Цена» и единица измерения значением из справочника товаров.
- После ввода поля «Количество» автоматически рассчитывается поле «Сумма»

3. Задача, возложенная на обработку «Работа с документами»:
Вывести в окно сообщений список всех видов документов, расположенных в конфигурации.
4. Настраиваемая особенность ведения учета в приложении:
Возможность исключения из документооборота ведения складского учета.

Вариант 2
1. Общий реквизит – Комментарий, тип строка, 100 символов.
2. Документ «Поступление товаров от поставщиков»
Поставщик_____
Склад_____

Товар Цена Количество Стоимость Ставка НДС Сумма НДС Всего с НДС

- При вводе нового документа поле «Склад» означивается по умолчанию выбранным значением из справочника складов.
- При выборе товара автоматически означивается поле «Цена» значением из справочника товаров.
- После ввода поля «Количество» автоматически рассчитывается поле «Сумма»

– Ставка НДС автоматически берется из соответствующей константы. Сумма НДС и Всего с НДС рассчитываются автоматически.

3. Задача, возложенная на обработку «Работа с документами»:
Вывести в окно сообщений даты всех документов разработанного вида за указанный период.

4. Настраиваемая особенность ведения учета в приложении:
Возможность исключения из документооборота ведения складского учета.

Вариант 3

1. Общий реквизит –Организация, тип «СправчочникСсылка.Организации»
2. Документ «Ввод начисленных сумм зарплаты »
% отчислений в ТФОМС _____
% отчислений ФФОМС _____
Подразделение_____
Сотрудник Сумма зарплаты Сумма отчислений Сумма отчислений
в ТФОМС ФФОМС

– Проценты отчислений автоматически берутся из соответствующих констант.
– Суммы отчислений рассчитываются после ввода суммы зарплаты.
– Задача, возложенная на обработку «Работа с документами»:

Пометить все введенные документы разработанного вида на удаление.

4. Настраиваемая особенность ведения учета в приложении: возможность исключения из документооборота ведения учета по подразделениям.

Вариант 4

1. Общий реквизит –ОтветственныйСотрудник, тип «Справчочник-Ссылка.Сотрудники»
2. Документ «Ввод начисленных сумм зарплаты »
Подразделение_____
Сотрудник Оклад Отработано дней по плану Оклад фактически Удержано НДФЛ Сумма «К выдаче»

- при вводе нового документа поле «Подразделение» означивается по умолчанию
- выбранным значением из справочника подразделений.
- Сумма планового оклада берется из справочника сотрудников
- Поля «Дней по плану», ставка НДФЛ берутся из соответствующих констант
- Поля «Оклад фактически», «Удержано НДФЛ», «Сумма к выдаче» рассчитываются после ввода поля «Отработано дней».

3. Задача, возложенная на обработку «Работа с документами»:
Отменить проведение всех введенных документов разработанного вида.
4**.** Настраиваемая особенность ведения учета в приложении: возможность исключения из документооборота ведения учета по подразделениям.

Вариант 5
1. Общий реквизит –ДатаСозданияДокумента, тип «Дата». Автоматически означивать при вводе нового документа текущей датой.
2. Документ « Поступление основных средств»
Подразделение_____
Поставщик_____

N Стр	Основное Средство	Первонач. Стоимость ОС	% износа	Сумма износа	Остаточная стоимость	Счет Затрат

- при вводе нового документа поля «Подразделение» и «Поставщик» означиваются по умолчанию выбранными значениями из соответствующих справочников.
- Поле «Счет затрат», берется из соответствующей константы по умолчанию
- Поля «Сумма износа», «Остаточная стоимость» рассчитываются после ввода поля «Первонач. Стоимость ОС» и процент износа.

3. Задача, возложенная на обработку «Работа с документами»:
Создать новый документ разработанного вида программным путем. Значения полей шапки документа брать произвольно. Сформировать две строки табличной части документа.
4. Настраиваемая особенность ведения учета в приложении: возможность исключения из документооборота ведения учета по подразделениям.

Порядок выполнения работы:
Составление алгоритма на языке программирования.
Написание программного кода на языке программирования.
Отладка программы на ПК.
Форма представления результата:
Алгоритм программы.
Программный код, составленный на языке программирования.

Тема 01.02.14 Регистры накопления. Макеты. Перечисления. Периодические регистры сведений

Практическое занятие № 45,46,47,48,49
Написание программ с использованием программы 1С:Предприятие

1. Добавление регистра накопления
2. Движение документа
3. Добавление отчета.
4. Макет печатной формы. Редактирование формы.
5. Добавление периодического регистра сведений

Цель работы: Написание программ с использованием регистров накопления. Научиться осуществлять движение документов.
Выполнив работу, Вы будете:
уметь:

- осуществлять разработку кода программного модуля на современных языках программирования;
- выполнять отладку и тестирование программы на уровне модуля;
- использовать инструментальные средства для автоматизации оформления документации;

Технические средства обучения:
персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением
Программное обеспечение:
Embarcadero RAD Studio 2010, 1С: Предприятие 8.3, GIMP, Git, MS Visual Studio 2010
Оборудование лаборатории и рабочих мест лаборатории:

- Пакет прикладных программ «Microsoft Office»: табличный процессор Microsoft Excel, текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer
- Компилятор tasm
- Отладчик машинных кодов AFDP
- MS Visual Studio 2010
- Embarcadero RAD Studio 2010
- 1С: Предприятие 8.3.

– Git.

Реализация рабочей программы ПМ предполагает обязательную учебную и производственную практику.

Оборудование и технологическое оснащение рабочих мест – Полигон учебных баз практики:

персональные компьютеры в локальной сети с доступом к сети Internet – 22 ед. с лицензионным программным и сетевым обеспечением;

локальная сеть с доступом к ресурсам интернет;

сервер

Программное обеспечение:

– Embarcadero RAD Studio 2010,
– 1С: Предприятие 8.3,
– GIMP,
– Git,
– MS Visual Studio 2010.

Задания

Источником данных для построения отчетов в СКД с нестандартной расшифровкой служат данные регистров накопления, полученные при проведении документов.

Задача состоит из следующих этапов.

– Создать новую подсистему. Все новые объекты, созданные в данной работе, поместить в данную подсистему.
– Разработать документы оперативного учета, необходимые для получения заданного отчета. Структуры документов разработать самостоятельно. Все расходные документы (реализация, списание, отпуск в производство) должны осуществлять контроль остатков. Стоимость списываемых товарно-материальных ценностей определять методом по среднему.
– Разработать регистры накопления, необходимые для хранения движений при проведении документов. Структуры регистров накопления разработать самостоятельно.
– Сформировать движения в регистрах накопления при проведении документов.
– Разработать отчет с использованием СКД. Структура отчета – см. варианты заданий.
– Разработать конкретизирующий отчет, который запускается автономно. Структуру конкретизирующего отчета разработать самостоятельно, исходя из целесообразной расшифровки основного отчета.
– Подключить конкретизирующий отчет в качестве нестандартной расшифровки основного отчета.

Вариант 1
Ведомость продажи товаров по заявкам клиентов за период с ___ по____

клиент	Всего товаров в заявках на сумму	Продано по заявкам клиентов на сумму	Осталось продать на сумму

Формируются два документа: документ “Заявка от покупателя» и документ «Реализация товаров», который может быть сформирован как на основании заявки, так и самостоятельно.

Вариант 2
Ведомость продажи товаров по заявкам клиента_____ за период с _____ по_____

Товар	Заявлено кол.	Сумма	Продано кол.	Сумма	Осталось продать кол.	сумма

Формируются два документа: “Заявка от покупателя» и «Реализация товаров», который может быть сформирован как на основании заявки, так и самостоятельно.

Вариант 3
Ведомость поступления товаров от поставщиков с учетов возвратов за период с __ по_____

Товар	Поступило Кол	Сумма	возвращено Кол	Сумма

Формируются два документа: “Поступление товаров» и «Возврат товаров».

Вариант 4
Ведомость поступления товаров от поставщиков за период с_ по ___

Поставщик	Товаров на сумму	НДС на сумму

Формируется документ: “Поступление товаров»

Вариант 5
Ведомость поступления товаров от поставщика _____ за ___ период С _____- по _____

Товар	Склад	Количество	цена	стоимость

Формируется документ: “Поступление товаров»

Порядок выполнения работы:
Составление алгоритма на языке программирования.
Написание программного кода на языке программирования.
Отладка программы на ПК.
Форма представления результата:
Алгоритм программы.
Программный код, составленный на языке программирования.

Тема 01.02.15 Перечисления. Оборотные регистры накопления. План видов характеристик.

Практическое занятие № 50,51,52,53,54
Написание программ с использованием программы 1С:Предприятие

1. Привязка номенклатуры к значениям перечисления.
2. Проведение документа по нескольким регистрам.
3. Добавление оборотного регистра накопления.
4. Работа с запросами.
5. Доработка объектов конфигурации. Создание характеристик

Цель работы: Написание программ с использованием оборотных регистров накопления. План видов характеристик.
Выполнив работу, Вы будете:
уметь:

- осуществлять разработку кода программного модуля на современных языках программирования;
- выполнять отладку и тестирование программы на уровне модуля;
- использовать инструментальные средства для автоматизации оформления документации;

Технические средства обучения:
персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением
Программное обеспечение:
Embarcadero RAD Studio 2010, 1С: Предприятие 8.3, GIMP, Git, MS Visual Studio 2010
Оборудование лаборатории и рабочих мест лаборатории:

- Пакет прикладных программ «Microsoft Office»: табличный процессор Microsoft Excel, текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer
- Компилятор tasm
- Отладчик машинных кодов AFDP
- MS Visual Studio 2010
- Embarcadero RAD Studio 2010
- 1С: Предприятие 8.3.

– Git.

Реализация рабочей программы ПМ предполагает обязательную учебную и производственную практику.

Оборудование и технологическое оснащение рабочих мест – Полигон учебных баз практики:

персональные компьютеры в локальной сети с доступом к сети Internet – 22 ед. с лицензионным программным и сетевым обеспечением;

локальная сеть с доступом к ресурсам интернет;

сервер

Программное обеспечение:

– Embarcadero RAD Studio 2010,
– 1С: Предприятие 8.3,
– GIMP,
– Git,
– MS Visual Studio 2010.

Задания

Создать новую подсистему. Все новые объекты, созданные в данной работе, поместить в данную подсистему.

– Разработать, если это необходимо, дополнительные документы.
– Разработать тип и структуру регистров накопления для хранения движений, формируемых при проведении документов.
– Разработать модули проведения документов.
– Разработать отчет. Средства обращения к источнику данных – запрос. Использовать конструктор запроса с обработкой результатов. Отчет должен выводить общие итоги.
– Отчет должен иметь расшифровку.
– В случае разработки нестандартного отчета вид отчета разработать самостоятельно.

Вариант 1

Ведомость поступления товаров от поставщиков за период с____ по____

Поставщик Товаров на сумму НДС на сумму Всего от поставщика на сумму Адрес поставщика

Разработать документ «Поступление товаров от поставщиков»

Расшифровка – нестандартная.

Вариант 2

Ведомость поступления товаров от поставщика _____ за ___ период

С _____ по _____

Товар Склад Единицы Колич. цена стоимость Сумма ндс Всего с ндс

Замечание 1. Строки отчета группируются по складам. В конце группы следует включать строку «Итого по складу».
Замечание 2 . Разработать документ «Поступление товаров от поставщиков» Расшифровка – нестандартная.

Вариант 3
Ведомость начисления зарплаты сотрудникам _____ за ___ период
С _____по _____

Подразделение	**Сотрудник**	**Сумма зарплаты**	**Сумма отчислений в ТФОМС**	**Сумма отчислений ФФОМС**

Замечание 1. Строки отчета группируются по подразделениям. В конце группы следует включать строку «Итого по подразделению».
Замечание 2. Разработать документ «Ввод начисленных сумм зарплаты »
Расшифровка – нестандартная.

Вариант 4
Ведомость поступления товаров на _____ склад от поставщика _____ за ___ период с _____ по ___

товар	**единицы**	**количество**	**цена**	**Стоимость**	**Сумма НДС**	**Всего с ндс**

Замечание 1. Строки отчета группируются по товарам. В конце группы следует включать строку «Итого товара».
Замечание 2. Разработать документ «Поступление товаров от поставщиков»
Расшифровка – нестандартная.

Вариант 5
Ведомость поступления товара _____ на склады от поставщиков
за ___ период с _____ по ___

Склад Документ Дата Поставщик Колич. Единицы Цена Стоимость

Замечание 1. Строки отчета группируются по поставщикам. В конце группы следует включать строку «Итого по поставщику».
Замечание 2. Разработать документ «Поступление товаров от поставщиков»
Расшифровка – стандартная.

Порядок выполнения работы:
Составление алгоритма на языке программирования.
Написание программного кода на языке программирования.

Отладка программы на ПК.

Форма представления результата:

Алгоритм программы.

Программный код, составленный на языке программирования.