МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(ДГТУ)
Колледж экономики, управления и права

Методические указания по организации практической работы студентов по учебной дисциплине Архитектура электронно-вычислительных машин и вычислительные системы

09.02.05 Прикладная информатика (по отраслям)

Ростов-на-Дону
2018

Содержание:

Практическая работа № 1

Тема: Перевод чисел из одной системы счисления в другую

Цель работы: научиться переводить целые числа, правильные и неправильные дроби из одной системы счисления в другую.

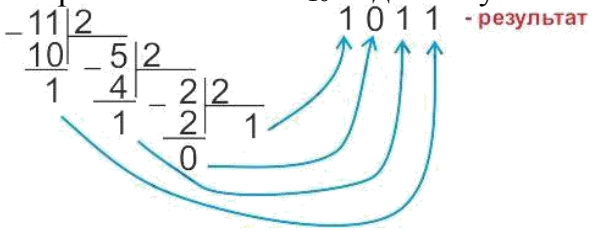
Системы счисления – совокупность приемов и правил записи чисел цифровыми знаками или символами. Системы счисления должны обеспечивать возможность представления любого числа в рассматриваемом диапазоне, единственность представления.

Десятичная система	Двоичная система	Шестнадцатеричная система
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C

13	1101	D
14	1110	E
15	1111	F
16	10000	10

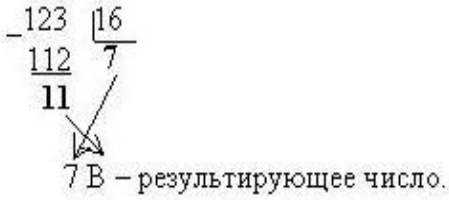
Перевод целых чисел.

Пример 1. Перевести число 11_{10} в двоичную систему счисления.



Ответ: 11_{10} □ 1011_2

Пример 2. Выполнить перевод числа 123 в шестнадцатеричную систему счисления.



Здесь остаток 11 преобразован в шестнадцатеричную цифру и после этого данная цифра вошла в число.

Пример 3. Выполнить перевод числа 13_{16} в десятичную систему счисления.

$13_{16} = 1*16^1 + 3*16^0 = 16 + 3 = 19.$

Таким образом, $13_{16} = 19.$

Перевод правильных дробей.

Пример 4. Перевести число $0{,}625_{10}$ в двоичную систему счисления.

Ответ: $0{,}625_{10}$ □ $0{,}101_2$

Пример 5. Выполнить перевод числа 0,847 в двоичную систему счисления.

Перевод выполнить до четырех значащих цифр после запятой.

Имеем

Таким образом, $0,847 = 0,1101_2$.

В данном примере процедура перевода прервана на четвертом шаге, поскольку получено требуемое число разрядов результата. Очевидно, это привело к потере ряда цифр.

Пример 6. Выполнить перевод из шестнадцатеричной системы счисления в десятичную число 0, $D8D_{16}$.

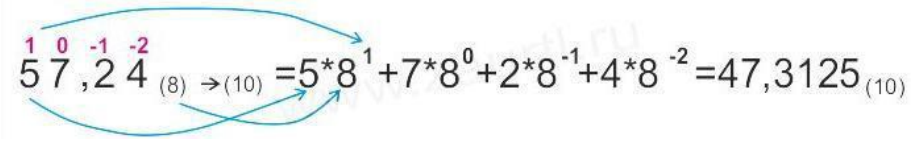
Имеем:

0, $D8D_{16}$ □13□$16^{□1}$ □ 8 □$16^{□2}$ □13□$16^{□3}$ □13□ 0,0625 □ 8 □0,003906 □13□ 0,000244 □ 0,84692

Таким образом, 0, $D8D_{16}$ □ 0,84692 .

Перевод неправильных дробей.

Пример 7. Перевести число $57,24_8$ в десятичную систему счисления.



Ответ: $57,24_8$ □ $47,3125_{10}$

Пример 8. Записать число $16,24_8$ в двоичной системе счисления.

Ответ: $16,24_8$ □ $1110,0101_2$

Методы работы: для проверки качества усвоенного материала студентам были предложены следующие вопросы и задания для самостоятельной работы.

Билет 1

1. Сколько и каких цифр используется в двоичной СС? Привести формулу полного значения числа 1011010_2.
2. Перевести число 674_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 111001_2 в десятичную СС. Перевести число 3251_8 в десятичную СС. Перевести число $3A7C_{16}$ в десятичную СС.
4. Перевести число 1010001_2 в шестнадцатеричную СС.
5. Перевести число $73F5_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,1_{10}$ в двоичную СС.

Билет 2

1. Сколько и каких цифр используется в восьмеричной СС? Привести формулу полного значения числа 6543_8.
2. Перевести число 960_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 101001_2 в десятичную СС. Перевести число 621_8 в десятичную СС. Перевести число $B1A_{16}$ в десятичную СС.
4. Перевести число 100001_2 в шестнадцатеричную СС.
5. Перевести число $6F16_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,2_{10}$ в двоичную СС.

Билет 3

1. Сколько и каких цифр используется в десятичной СС? Привести формулу полного значения числа 6543_{10}.
2. Перевести число 581_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 101101_2 в десятичную СС. Перевести число 123_8 в десятичную СС. Перевести число $26E_{16}$ в десятичную СС.
4. Перевести число 1111100_2 в шестнадцатеричную СС.
5. Перевести число $1A34_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,3_{10}$ в двоичную СС.

Билет 4

1. Сколько и каких цифр используется в шестнадцатеричной СС? Привести формулу полного значения числа $102A_{16}$.
2. Перевести число 951_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 1011110_2 в десятичную СС. Перевести число 234_8 в десятичную СС. Перевести число $5D1_{16}$ в десятичную СС.
4. Перевести число 1000111100_2 в шестнадцатеричную СС.
5. Перевести число $25F3_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,4_{10}$ в двоичную СС.

Билет 5

1. Сколько и каких цифр используется в троичной СС? Привести формулу полного значения числа 102_3.
2. Перевести число 247_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 101010_2 в десятичную СС. Перевести число 357_8 в десятичную СС. Перевести число $8CA_{16}$ в десятичную СС.
4. Перевести число 1111000110_2 в шестнадцатеричную СС.

5. Перевести число $9BA56_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,5_{10}$ в двоичную СС.

Билет 6

1. Сколько и каких цифр используется в шестеричной СС? Привести формулу полного значения числа 452_6.
2. Перевести число 957_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 00011110_2 в десятичную СС. Перевести число 731_8 в десятичную СС. Перевести число 789_{16} в десятичную СС.
4. Перевести число 100111000_2 в шестнадцатеричную СС.
5. Перевести число $1234B_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,6_{10}$ в двоичную СС.

Билет 7

1. Сколько и каких цифр используется в пятеричной СС? Привести формулу полного значения числа 432_5.
2. Перевести число 100_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 11110001_2 в десятичную СС. Перевести число 654_8 в десятичную СС. Перевести число ABC_{16} в десятичную СС.
4. Перевести число 110011001100_2 в шестнадцатеричную СС.
5. Перевести число $3456F_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,7_{10}$ в двоичную СС.

Билет 8

1. Сколько и каких цифр используется в девятерной СС? Привести формулу полного значения числа 432_9.
2. Перевести число 105_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 10001110_2 в десятичную СС. Перевести число 777_8 в десятичную СС. Перевести число 100_{16} в десятичную СС.
4. Перевести число 000111000111000_2 в шестнадцатеричную СС.
5. Перевести число $98AB_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,8_{10}$ в двоичную СС.

Билет 9

1. Сколько и каких цифр используется в двоичной СС? Привести формулу полного значения числа 100_2.
2. Перевести число 200_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 11110000_2 в десятичную СС. Перевести число 654_8 в десятичную СС. Перевести число 500_{16} в десятичную СС.
4. Перевести число 111101010101_2 в шестнадцатеричную СС.
5. Перевести число $222C_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,9_{10}$ в двоичную СС.

Билет 10

1. Сколько и каких цифр используется в десятичной СС? Привести формулу полного значения числа 165_{10}.
2. Перевести число 300_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 1000100000_2 в десятичную СС. Перевести число 532_8 в десятичную СС. Перевести число $11AB_{16}$ в десятичную СС.
4. Перевести число 100000111111001_2 в шестнадцатеричную СС.
5. Перевести число $876FD_{16}$ в двоичную СС.

6. Перевести правильную дробь $0,1_{10}$ в двоичную СС.

Билет 11

1. Сколько и каких цифр используется в шестнадцатеричной СС? Привести формулу полного значения числа 165_{16}.
2. Перевести число 509_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 100001_2 в десятичную СС. Перевести число 641_8 в десятичную СС. Перевести число $234E_{16}$ в десятичную СС.
4. Перевести число 111110011001100_2 в шестнадцатеричную СС.
5. Перевести число $123AB_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,2_{10}$ в двоичную СС.

Билет 12

1. Сколько и каких цифр используется в семеричной СС? Привести формулу полного значения числа 165_7.
2. Перевести число 561_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 101101_2 в десятичную СС. Перевести число 701_8 в десятичную СС. Перевести число $A63E_{16}$ в десятичную СС.
4. Перевести число 1100100111101101_2 в шестнадцатеричную СС.
5. Перевести число $AB1CD_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,3_{10}$ в двоичную СС.

Билет 13

1. Сколько и каких цифр используется в пятеричной СС? Привести формулу полного значения числа 142_5.
2. Перевести число 567_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 01110000_2 в десятичную СС. Перевести число 333_8 в десятичную СС. Перевести число $3A4B_{16}$ в десятичную СС.
4. Перевести число 011000111110000_2 в шестнадцатеричную СС.
5. Перевести число $5F3E_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,4_{10}$ в двоичную СС.

Билет 14

1. Сколько и каких цифр используется в шестнадцатеричной СС? Привести формулу полного значения числа 142_{16}.
2. Перевести число 831_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 0111001_2 в десятичную СС. Перевести число 345_8 в десятичную СС. Перевести число AAA_{16} в десятичную СС.
4. Перевести число 11001100110011000_2 в шестнадцатеричную СС.
5. Перевести число $7653E_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,5_{10}$ в двоичную СС.

Билет 15

1. Сколько и каких цифр используется в двоичной СС? Привести формулу полного значения числа 1010_2.
2. Перевести число 320_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 11000111_2 в десятичную СС. Перевести число 222_8 в десятичную СС. Перевести число $ABC3_{16}$ в десятичную СС.
4. Перевести число 11000011110000_2 в шестнадцатеричную СС.
5. Перевести число $160AFE_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,6_{10}$ в двоичную СС.

Билет 16

1. Сколько и каких цифр используется в троичной СС? Привести формулу полного значения числа 1010_3.
2. Перевести число 702_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 100001_2 в десятичную СС. Перевести число 732_8 в десятичную СС. Перевести число 1237_{16} в десятичную СС.
4. Перевести число 1100001111111011_2 в шестнадцатеричную СС.
5. Перевести число $674B_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,7_{10}$ в двоичную СС.

Билет 17

1. Сколько и каких цифр используется в шестеричной СС? Привести формулу полного значения числа 1234_6.
2. Перевести число 800_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 1001100_2 в десятичную СС. Перевести число 653_8 в десятичную СС. Перевести число $675B_{16}$ в десятичную СС.
4. Перевести число 1100111100011001_2 в шестнадцатеричную СС.
5. Перевести число $FE34_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,8_{10}$ в двоичную СС.

Билет 18

1. Сколько и каких цифр используется в десятичной СС? Привести формулу полного значения числа 1234_{10}.
2. Перевести число 789_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 1011001_2 в десятичную СС. Перевести число 657_8 в десятичную СС. Перевести число $504A_{16}$ в десятичную СС.
4. Перевести число $10001001101010111100110 1_2$ в шестнадцатеричную СС.
5. Перевести число 5661_{16} в двоичную СС.
6. Перевести правильную дробь $0,9_{10}$ в двоичную СС.

Билет 19

1. Сколько и каких цифр используется в шестнадцатеричной СС? Привести формулу полного значения числа 1234_{16}.
2. Перевести число 450_{10} в двоичную, восьмеричную и шестнадцатеричные СС.
3. Перевести число 10100011_2 в десятичную СС. Перевести число 700_8 в десятичную СС. Перевести число $345E_{16}$ в десятичную СС.
4. Перевести число 100000111110001111_2 в шестнадцатеричную СС.
5. Перевести число $98AB_{16}$ в двоичную СС.
6. Перевести правильную дробь $0,1_{10}$ в двоичную СС.

Практическая работа № 2.

Тема: Использование обратного и дополнительного двоичных кодов для реализации всех арифметических операций с помощью суммирующего устройства.

Цель работы: Научиться производить операции сложения и вычитания в дополнительных и обратных кодах

1.Краткие теоретические сведения:

Особенности сложения чисел в обратном и дополнительном кодах. При сложении чисел в дополнительном коде возникающая единица переноса в знаковом разряде отбрасывается. При сложении чисел в обратном коде возникающая единица переноса в знаковом разряде прибавляется к младшему разряду суммы кодов. Если результат арифметических действий является кодом отрицательного числа, необходимо преобразовать его в прямой код. При этом обратный код преобразуется в прямой заменой цифр во всех разрядах кроме знакового на противоположные. Дополнительный код преобразуется в прямой также, как и обратный, с последующим прибавлением единицы к младшему разряду.

Пример Сложить двоичные числа X и Y в обратном и дополнительном кодах.

А) X= 111, Y= -11;

1) Сложим числа, пользуясь правилами двоичной арифметики:

$$\begin{array}{rr} X= & 111 \\ Y= & \underline{\ 11} \\ X+Y= & 100 \end{array}$$

2) Сложим числа, используя коды:

Прямой код	Сложение в обратном коде	Сложение в дополнительном коде
$X_{пр}$=0,0000111 $Y_{пр}$=1,0000011	$X_{обр}$= 0,0000111 $Y_{обр}$= + 1,1111100 1 0,0000011 +1 $(X+Y)_{обр}$= 0,0000100	$X_{доп}$= 0,0000111 $Y_{доп}$= + 1,1111101 1)0,0000100 отбрасывается $(X+Y)_{доп}$= 0,0000100

Так как результат сложения является кодом положительного числа (знак 0), то (X+Y)обр=(X+Y)доп=(X+Y)пр

.) X= -101,Y= -11;

1) Сложим числа, пользуясь правилами двоичной арифметики:

$$\begin{aligned} X &= -\ 101 \\ Y &= -\ 110 \\ X+Y &= -1011 \end{aligned}$$

2) Сложим числа, используя коды:

Прямой код	Сложение в обратном коде	Сложение в дополнительном коде
$X_{пр}$=1,0000101 $Y_{пр}$=1,0000110	$X_{обр}$= 1,1111010 $Y_{обр}$= + 1,1111001 1 1,1110011 +1 $(X+Y)_{обр}$= 1,1110100	$X_{доп}$= 1,1111011 $Y_{доп}$= + 1,1111010 1)1,1110101 отбрасывается $(X+Y)_{доп}$= 1,1110101

Так как сумма является кодом отрицательного числа (знак 1), то необходимо перевести результаты в прямой код:

- из обратного кода (X+Y)обр=1,1110100 (X+Y)пр=1,0001011;

- из дополнительного кода (X+Y)доп=1,1110101 (X+Y)пр=1,0001010+0,0000001=1,0001011. Таким образом, X+Y= -1011 и полученный результат совпадает с обычной записью

2.Задания для самостоятельного выполнения.

Задание 1. Вычислить выражение $-3_{(10)}$ $-2_{(10)}$.в прямом и обратном коде.

Задание 2.Вычислить выражение $7_{(10)}$ $-3_{(10)}$.в прямом и обратном коде.

Задание 3.Число $-5_{(10)}$ перевести в дополнительный код и обратно.

Задание 4Число $-44_{(10)}$ ($10101100_{(2)}$) перевести в дополнительный код и обратно Задание 5.Перевести в дополнительный код модуль числа -44.

Задание 6.Вычислить алгебраическую сумму 26 – 34. (с использованием обратного кода).

Задание 7.Перевести число 1110 из дополнительного кода в десятичную систему.

Практическая работа № 3

Тема: «Изучение основных логических функций и принципов работы логических элементов»

Цель работы: изучить основные логические функции и принципы работы логических элементов.

История логики насчитывает около двух с половиной тысячелетий. Она берет начало от формальной логики Аристотеля. Дальнейшее развитие логики связано с именами Готфрида Вильгельма Лейбница и Джорджа Буля, которые разработали математический аппарат алгебры логики. Именно поэтому алгебру высказываний и называют булевой алгеброй. Математическая логика в XIX—XX вв. рассматривалась как базис для логического обоснования в различных областях математики, но в последние десятилетия математическая логика находит применение во многих областях, в частности, в кибернетике, теории ЭВМ, теории алгоритмов.

Основным понятием математической логики является понятие высказывания. Высказывание — предложение, про которое всегда можно сказать, истинно оно или ложно. Высказывания бывают простые и сложные. Сложное высказывание состоит из простых, соединенных знаками логических операций. Простые высказывания обычно обозначают большими латинскими буквами: A, B, C, — и т.д.

Рассмотрим следующие логические операции:
не(отрицание),или(конъюнкция), и(дизъюнкция),→(следование).

Логические операции определяются через таблицы истинности.

Пусть имеются два высказывания: A — юноша в школе; B — юноша на уроке. Составим таблицы, которые показывают смысл операций и, или, не.

Операция “и”		
A	B	A и B
И	И	И
И	Л	Л
Л	И	Л
Л	Л	Л

Операция “или”		
A	B	A и B
И	И	И
И	Л	И
Л	И	И

Л	Л	Л

Операция “не”	
A	не A
И	Л
Л	И

Пусть имеются два высказывания: А — у человека высокая температура, В — человек болен. Составим таблицу, которая показывает смысл операции следования.

А	В	Из А следует В
И	И	И
И	Л	Л
Л	И	И
Л	Л	И

Решение задач.

Дана таблица:

Формула	Высказывание	Тигр	Волк	Бурундук	Заяц
А	Зверь полосатый				
В	Зверь хищный				
не А					
не В					
А и В					
А или В					

Ответ:

Формула	Высказывание	Тигр	Волк	Бурундук	Заяц
А	Зверь полосатый	И	Л	И	Л
В	Зверь хищный	И	И	Л	Л
не А	Зверь не полосатый	Л	И	Л	И
не В	Зверь не хищный	Л	Л	И	И
А и В	Зверь полосатый и хищный	И	Л	Л	Л
А или В	Зверь полосатый или хищный	И	И	И	И

Вопросы для самоконтроля

Операция конъюнкции. Таблица истинности.

Операция дизъюнкции. Таблица истинности.

Операция отрицания. Таблица истинности.

Выполните следующие задания

1. Пользуясь формулой, построить схему комбинационного устройства.

f = aΛbVaΛb

f = aVbΛ(bVa)Λa

f = aΛ(aVb)V(bΛa)

Практическая работа № 4.

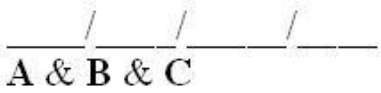
Тема: Построение элементарных логических схем: триггеры, сумматоры, шифраторы, дешифраторы.

Цель работы: изучение работы основных логических элементов ЭВМ.

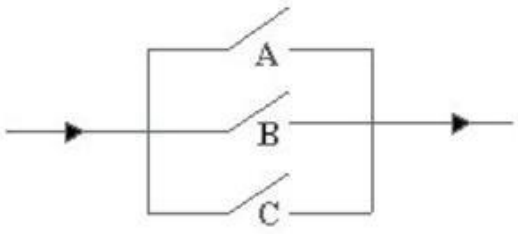
1.Краткие теоретические сведения.

Знания из области математической логики можно использовать для конструирования электронных устройств. Нам известно, что 0 и 1 в логике не просто цифры, а обозначение состояний какого-то предмета нашего мира, условно называемых «ложь» и «истина». Таким предметом, имеющим два фиксированных состояния, может быть электрический ток. Устройства, фиксирующие два устойчивых состояния, называются бистабильными (например, выключатель, реле). Если вы помните, первые вычислительные машины были релейными. Позднее были созданы новые устройства управления электричеством – электронные схемы, состоящие из набора полупроводниковых элементов. Такие электронные схемы, которые преобразовывают сигналы только двух фиксированных напряжений электрического тока (бистабильные), стали называть *логическими элементами*.

На элементарном уровне конъюнкцию можно представить себе в виде последовательно соединенных выключателей, а дизъюнкцию – в виде параллельно соединенных выключателей:



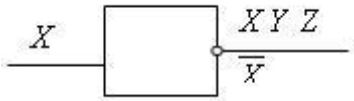
A & B & C



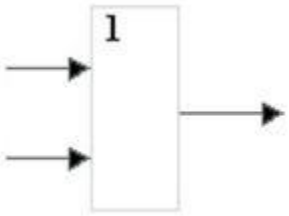
A V B V C

Логические элементы имеют один или несколько входов и один выход, через которые проходят электрические сигналы, обозначаемые условно 0, если «отсутствует» электрический сигнал, и 1, если «имеется» электрический сигнал.

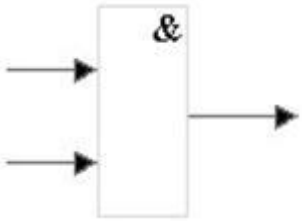
Простейшим логическим элементом является *инвертор*, выполняющий функцию отрицания. Если на вход поступает сигнал, соответствующий 1, то на выходе будет 0. И наоборот. У этого элемента один вход и один выход. На функциональных схемах он обозначается:



Логический элемент, выполняющий логическое сложение, называется *дизъюнктор*. Он имеет, как минимум, два входа. На функциональных схемах он обозначается:

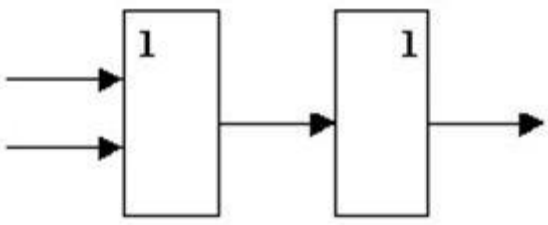


Логический элемент, выполняющий логическое умножение, называется конъюнктор. Он имеет, как минимум, два входа. На функциональных схемах он обозначается:



Специальных логических элементов для импликации и эквивалентности нет, т.к. A => B можно заменить на мA V B ; A ⇔ B можно заменить на (A & B)V(мA&мB).

Другие логические элементы построены из этих трех простейших и выполняют более сложные логические преобразования информации. Сигнал, выработанный одним логическим элементом, можно подавать на вход другого элемента, это дает возможность образовывать цепочки из отдельных логических элементов. Например:

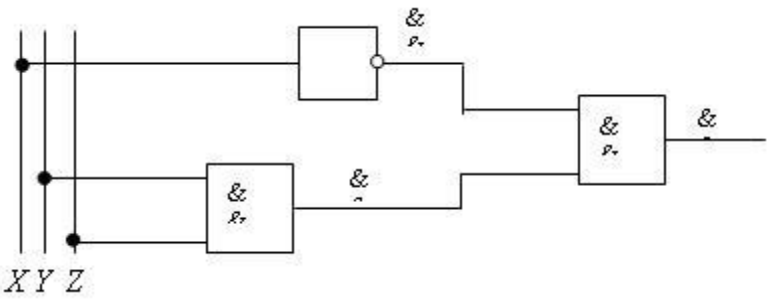


Эта схема соответствует сложной логической функции F(A,B)= м (A V B).

Попробуйте проследить изменения электрического сигнала в этой схеме. Например, какое значение электрического сигнала (0 или 1) будет на выходе, если на входе: A=1 и B=0.

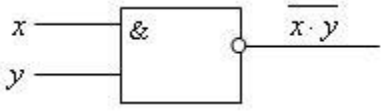
Такие цепи из логических элементов называются *логическими устройствами*. Логические устройства же, соединяясь, в свою очередь образуют *функциональные схемы* (их еще называют структурными или *логическими схемами*). По заданной функциональной схеме можно определить логическую формулу, по которой эта схема работает, и наоборот.

Пример **.** Логическая схема для функции $F(X,Y,Z)=\overline{X}\,\&\,(Y \vee Z)$ будет выглядеть следующим образом:



Правила составления электронных логических схем по заданным таблицам истинности остаются такими же, как для контактных схем.

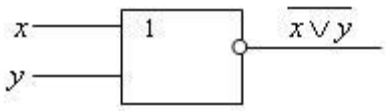
Рассмотрим еще два логических элемента, которые играют роль базовых при создании более сложных элементов и схем.



Логический элемент И-НЕ состоит из конъюнктора и инвертора:

Выходная функция выражается формулой $F(x,y)=\overline{xy}$.

Логический элемент ИЛИ-НЕ состоит из дизъюнктора и инвертора:



Выходная функция выражается формулой $F(x,y) = \overline{x \vee y}$.

Логическая реализация типовых устройств компьютера

Обработка любой информации на компьютере сводится к выполнению процессором различных арифметических и логических операций. Для этого в состав процессора входит так называемое арифметико-логическое устройство (АЛУ). Оно состоит из ряда устройств, построенных на рассмотренных выше логических элементах. Важнейшими из таких устройств являются *триггеры, полусумматоры, сумматоры, шифраторы, дешифраторы, счетчики, регистры*.

Этапы конструирования логического устройства.

Конструирование логического устройства состоит из следующих этапов:

1. Построение таблицы истинности по заданным условиям работы проектируемого узла (т.е. по соответствию его входных и выходных сигналов).
2. Конструирование логической функции данного узла по таблице истинности, ее преобразование (упрощение), если это возможно и необходимо.
3. Составление функциональной схемы проектируемого узла по формуле логической функции. После этого остается только реализовать полученную схему.

Попробуем, действуя по этому плану, сконструировать устройство для сложения двух
двоичных чисел (одноразрядный полусумматор).

Пусть нам необходимо сложить двоичные числа X и Y. Через P и Z обозначим первую и вторую цифру суммы: X + Y = PZ. Вспомните таблицу сложения двоичных чисел.

1. Таблица истинности, определяющая результат сложения, имеет вид:

X	Y	P	Z
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

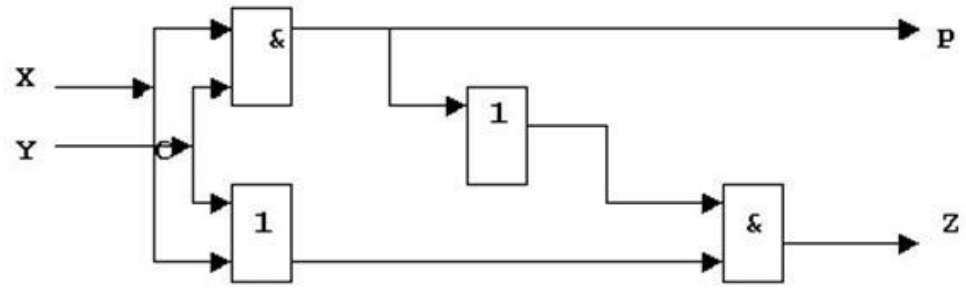
2. Сконструируем функции P(X,Y) и Z(X,Y) по этой таблице: P(X,Y)=X & Y; Z(X,Y)=(м X&Y)V (X& м Y).

Преобразуем вторую формулу, пользуясь законами логики.

(м X&Y)V(X&мY) ⇔ ((м X&Y)VX)& ((м X&Y)V мY) ⇔ (XV(м X&Y))&(м YV (Y&мX))

⇔((XV мX)&(XVY))& ((м YVY)&(м Y V мX)) ⇔ (1&(XVY))&((И& м(X&Y)) ⇔ (XVY)& м(X&Y).

3. Теперь можно построить функциональную схему одноразрядного полусумматора:



Одноразрядный двоичный сумматор на три входа и два выхода называется *полным одноразрядным сумматором*.

Сумматор – это электронная логическая схема, выполняющая суммирование двоичных чисел поразрядным сложением. Сумматор является центральным узлом арифметико-логического устройства процессора. Находит он применение и в других устройствах компьютера. Сумматор выполняет сложение *многозначных двоичных чисел*. Он представляет собой последовательное соединение *одноразрядных двоичных сумматоров*, каждый из которых осуществляет сложение в одном разряде. Если при этом возникает переполнение разряда, то перенос суммируется с содержимым старшего соседнего разряда.

Общая схема сумматора:

Триггер – электронная схема, применяемая для хранения значения одноразрядного двоичного кода.

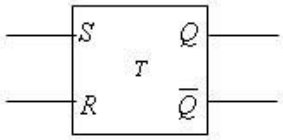
Воздействуя на входы триггера, его переводят в одно из двух возможных состояний (0 или 1). С поступлением сигналов на входы триггера в зависимости от его состояния либо происходит переключение, либо исходное состояние

сохраняется. При отсутствии входных сигналов триггер сохраняет свое состояние сколь угодно долго.

Термин *триггер* происходит от английского слова *trigger* – защёлка, спусковой крючок. Для обозначения этой схемы в английском языке чаще употребляется термин *flip-flop*, что в переводе означает «хлопанье». Это звукоподражательное название электронной схемы указывает на её способность почти мгновенно переходить («перебрасываться») из одного электрического состояния в другое.

Существуют разные варианты исполнения триггеров в зависимости от элементной базы (И-НЕ, ИЛИ-НЕ) и функциональных связей между сигналами на входах и выходах (*RS, JK, T, D* и другие).

Самый распространённый тип триггера – это *RS*-триггер (*S* и *R* соответственно от английских *set* – установка, и *reset* – сброс). Условное обозначение *RS*-триггера:

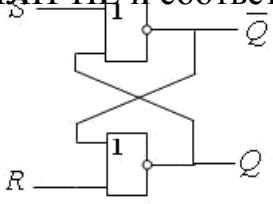


Триггер имеет два симметричных входа *S* и *R*, которые используются для установки в единичное состояние и сброса, - в нулевое. Еще у него есть два симметричных выхода *Q* и $\overline{Q}$, причем выходной сигнал *Q* является логическим отрицанием сигнала $\overline{Q}$.

За единичное состояние триггера условились принимать такое, при котором *Q=1*.

На каждый из входов *S* и *R* могут подаваться входные сигналы в виде кратковременных импульсов  . Наличие импульса на входе считается единицей, а его отсутствие – нулем. Ниже показана схема реализации триггера с помощью элементов ИЛИ-НЕ и соответствующая таблица истинности.

S	R	Q	$\overline{Q}$	Режим триггера
1	0	1	0	Установка 1
0	1	0	1	Установка 0
0	0	Последние значения		Хранение информации
1	1	Запрещено		



Два одинаковых двухвходовых логических элемента ИЛИ-НЕ соединены симметричным образом. Сигнал, поданный на один из входов каждого элемента, снимается с выхода другого. Наличие такого соединения и дает триггеру возможность сохранять свое состояние после прекращения действия сигналов (никакой другой логический элемент не в состоянии поддерживать сигнал на выходе после прекращения действия входного напряжения).

Проанализируем возможные комбинации значений входов *R* и *S* триггера, используя его схему и таблицу истинности схемы.

1. Пусть поданы сигналы *S=1, R=0*. Независимо от состояния другого входа на выходе верхнего элемента появится 0. Этот нулевой сигнал подается на вход нижнего элемента, где уже есть *R=0*. Выход нижнего элемента станет равным 1. Эта единица возвращается на вход первого элемента. Теперь состояние другого входа *(S)* этого элемента роли не играет: если даже убрать входной сигнал *S*, состояние триггера останется без изменения. Поскольку *Q=1*, триггер перешел в единичное состояние, устойчивое, пока не придут новые внешние сигналы.
2. При *S=0* и *R=1* вследствие симметричности схемы все происходит аналогично, но теперь на выходе Q будет 0. То есть при подаче сигнала на вход *R* триггер сбрасывается в устойчивое нулевое состояние.
3. При окончании действия обоих сигналов (*R=0* и *S=0*) триггер сохраняет на выходе *Q* тот сигнал, который был установлен входным импульсом *S* или *R*.

4. Подача импульсов одновременно на входы R и S может привести к неоднозначному результату, поэтому эта комбинация входных сигналов (*R=1* и *S=1*) запрещена.

Один триггер хранит один бит информации. Для хранения одного байта информации необходимо 8 триггеров. Современные микросхемы памяти содержат миллионы триггеров.

По технологии изготовления память делится на статическую и динамическую. На триггерах основана статическая память, а динамическая устроена по принципу конденсатора: заряженный конденсатор соответствует единице, а незаряженный – нулю.

Динамическая память проще по устройству, имеет больший объем и дешевле. В силу этих преимуществ в настоящее время основной объем оперативного запоминающего устройства компьютера является динамическим.

Однако статическая память имеет более высокое быстродействие. Кэш-память имеет статическую природу, что позволяет согласовать высокое быстродействие процессора и низкую скорость работы динамической памяти.

Конденсаторы динамической памяти постепенно разряжаются через внешние цепи, и потому требуют периодической подзарядки, чтобы не потерять информацию. Этот процесс называется регенерацией памяти, его наличие усложняет подключение микросхем динамической памяти. Микросхема статической памяти сильнее нагревается при работе, так как использует активные элементы – транзисторы.

Некоторое количество триггеров, объединенных вместе общей системой управления, называется *регистром*. Регистры содержатся в различных вычислительных узлах компьютера – процессоре, периферийных устройствах и т.д. Регистр – это устройство, предназначенное для хранения многоразрядного двоичного числового кода, которым можно представлять и адрес, и команду, и данные.

Упрощенно регистр можно представить как совокупность ячеек, в каждой из которых может быть записано одно из двух значений: 0 или 1, то есть один разряд двоичного числа.

Существует несколько типов регистров, отличающихся видом выполняемых операций.

Некоторые важные регистры имеют свои названия, например:

сдвиговый регистр – предназначен для выполнения операции сдвига; *счетчики* – схемы, способные считать поступающие на вход импульсы. К ним относятся *T*- триггеры (название от англ. *Tumble* – опрокидываться). Этот триггер имеет один счетный вход и два выхода. Под действием сигналов триггер меняет свое состояние с нулевого на единичное и наоборот. Число перебрасываний соответствует числу поступивших сигналов;

счетчик команд – регистр устройства управления процессора (УУ), содержимое которого соответствует адресу очередной выполняемой команды; служит для автоматической выборки программы из последовательных ячеек памяти;

регистр команд – регистр УУ для хранения кода команды на период времени, необходимый для ее выполнения. Часть его разрядов используется для хранения кода операции, остальные – для хранения кодов адресов операндов.

Техническая сторона логики компьютера основана на технологии транзистора, что позволяет получать одну из двух возможных единиц информации (0 и 1), оперируя с передачей или отсутствием передачи тока. На следующем уровне вступает система носителей (переносчиков) информации – это нули и единицы (0 и 1), которые отражают в себе реальную информацию путем применения, как систем счисления, так и системы команд микропроцессора.

Логика микропроцессора (МП). МП имеет (может иметь) встроенную логику, основанную на формальной логике человека. Таким образом, имеется возможность обрабатывать информацию. Так, например, получая группу информационных потоков (два потока), МП может, используя формальную логику, выдать один искомый поток.

Логика операционной системы. Хотя технически возможна работа на компьютере без использования операционной системы (ОС), так как в ПЗУ находятся (могут находиться) для этого специальные программы, целесообразность использования ОС никем не подвергается сомнению. ОС представляет собой программу (группу программ), которая обеспечивает полный системный интерфейс компьютера.

И, наконец, логика прикладных программ. В данном случае все зависит от фантазии и профессионализма программиста или пользователя.

Проиллюстрируем уровни логики ЭВМ. Получив на два регистра по единице и команду

реализовать функцию AND, МП, применив заложенную в нем формальную логику, выдает на результирующий регистр единицу. Эта единица некоторым образом интерпретируется операционной системой, например, как истинность выполненной операции, а затем передается (может передаваться) прикладной программе, которая в свою очередь так же интерпретирует полученную информацию и производит в соответствии с этим некоторые действия.

2.Задания для самостоятельной работы.

Задание 1. По заданной логической функции F(A,B) = B&¬AV ¬B&A построить логическую схему.

Задание 2.Логическая схема имеет два входа X и Y. Определить логические функции F1(X,Y) и F2 (X,Y), которые реализуются на ее двух выходах.

Задание 3.Построить таблицу истинности одноразрядного двоичного сумматора, воспользовавшись таблицей сложения двоичных чисел.

Задание 4.Построить таблицу, описывающую состояние входов и выходов RS-триггера.

Задание 5. Построить схему линейного шифратора на 8 входов.

Практическая работа № 5-6

Тема: Архитектура персонального компьютера.

Цель работы: изучить архитектуру ПК, основные и дополнительные устройства ПК, их назначение и взаимосвязь.

Основная компоновка частей компьютера и связь между ними называется **архитектурой**. При описании архитектуры компьютера определяется состав входящих в него компонент, принципы их взаимодействия, а также их функции и характеристики.

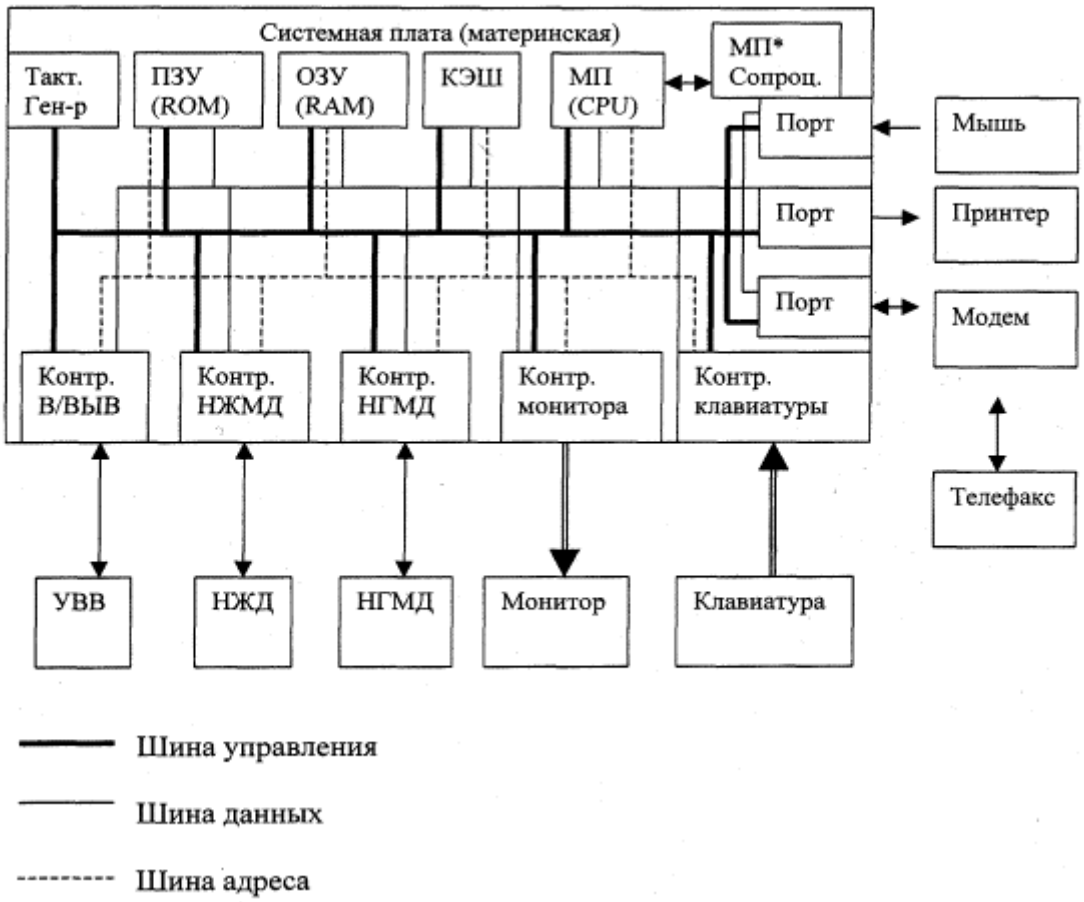


Рис. 1 Архитектура персонального компьютера

Практически все универсальные ЭВМ отражают классическую неймановскую архитектуру, представленную на схеме. Эта схема во многом характерна как для микроЭВМ, так и для мини ЭВМ и ЭВМ общего назначения.

Теоретические сведения:

Рассмотрим устройства подробнее.

Основная часть системной платы — **микропроцессор (МП)** или CPU (Central Processing Unit), он управляет работой всех узлов ПК и программой, описывающей алгоритм решаемой

задачи. МП имеет сложную структуру в виде электронных логических схем. В качестве его компонент можно выделить:

А). АЛУ - арифметико-логическое устройство, предназначенное для выполнения арифметических и логических операций над данными и адресами памяти;

Б). Регистры или микропроцессорная память — сверхоперативная память, работающая со скоростью процессора, АЛУ работает именно с ними;

В). УУ - устройство управления - управление работой всех узлов МП посредством выработки и передачи другим его компонентам управляющих импульсов, поступающих от кварцевого тактового генератора, который при включении ПК начинает вибрировать с постоянной частотой (100 МГц, 200-400 МГц). Эти колебания и задают темп работы всей системной платы;

Г). СПр - система прерываний - специальный регистр, описывающий состояние МП, позволяющий прерывать работу МП в любой момент времени для немедленной обработки некоторого поступившего запроса, или постановки его в очередь; после обработки запроса СПр обеспечивает восстановление прерванного процесса;

Д). Устройство управления общей шиной — интерфейсная система.

Для расширения возможностей ПК и повышения функциональных характеристик микропроцессора дополнительно может поставляться математический сопроцессор, служащий для расширения набора команд МП. Например, математический сопроцессор IBM-совместимых ПК расширяет возможности МП для вычислений с плавающей точкой; сопроцессор в локальных сетях (LAN-процессор) расширяет функции МП в локальных сетях.

Характеристики процессора:

быстродействие (производительность, тактовая частота) — количество операций, выполняемых в секунду.

разрядность — максимальное количество разрядов двоичного числа, над которыми одновременно может выполняться машинная операция.

Пример 2.5.1. *Первый процессор был 4-разрядным, то есть работал с числами, представляемыми 4 двоичными разрядами - 2'*= 16 чисел, 16 адресов.*

16-разрядный процессор одновременно может работать с 2^{16}=б5536 числами и адресами. 32-разрядный - 2^{32}=4 294 967 296.чисел.

При тактовой частоте 33 МГц обеспечивается выполнение 7 млн. коротких машинных операций (+,, пересылка информации); при частоте 100 МГц -20 млн. аналогичных операций.*

Интерфейсная система - это:

-шина управления (ШУ) - предназначена для передачи управляющий импульсов и синхронизации сигналов ко всем устройствам ПК;

-шина адреса (ША) - предназначена для передачи кода адреса ячейки памяти или порта ввода/вывода внешнего устройства;

-шина данных (ШД) - предназначена для параллельной передачи всех разрядов числового кода;

-шина питания - для подключения всех блоков ПК к системе электропитания.

Интерфейсная система обеспечивает три направления передачи информации:

- между МП и оперативной памятью;
- между МП и портами ввода/вывода внешних устройств;
- между оперативной памятью и портами ввода/вывода внешних устройств. Обмен информацией между устройствами и системной шиной происходит с помощью кодов ASCII.

Память - устройство для хранения информации в виде данных и программ. Память делится прежде всего на внутреннюю (расположенную на системной плате) и внешнюю (размещенную на разнообразных внешних носителях информации).

Внутренняя память в свою очередь подразделяется на:

- **ПЗУ** (постоянное запоминающее устройство) или ROM (read only memory), которое содержит - постоянную информацию, сохраняемую даже при отключенном питании, которая служит для тестирования памяти и оборудования компьютера, начальной загрузки ПК при включении. Запись на специальную кассету ПЗУ происходит на заводе фирмы-изготовителя ПК и несет черты его индивидуальности. **Объем** ПЗУ относительно невелик - от 64 до 256 Кб.

- **ОЗУ** (оперативное запоминающее устройство, ОП — оперативная память) или RAM (random access memory), служит для оперативного хранения программ и данных, сохраняемых только на период работы ПК. Она энергозависима, при отключении питания информация теряется. ОП выделяется особыми функциями и спецификой доступа:

(1) ОП хранит не только данные, но и выполняемую программу;

(2) МП имеет возможность прямого доступа в ОП, минуя систему ввода/вывода.

Логическая организация памяти — адресация, размещение данных определяется ПО, установленным серый гей на ПК, а именно ОС.

Объем ОП колеблется в пределах от 64 Кб до 64 Мб и выше, как правило, ОП имеет модульную структуру и может расширяться за счет добавления новых микросхем.

Кэш-память - имеет малое время доступа, служит для временного хранения промежуточных результатов и содержимого наиболее часто используемых ячеек ОП и регистров МП.

Объем кэш-памяти зависит от модели ПК и составляет обычно 256 Кб.

Внешняя память. Устройства внешней памяти весьма разнообразны. Предлагаемая классификация учитывает тип **носителя**, т.е. материального объекта, способного хранить информацию.

(1) **Накопители на магнитной ленте** исторически появились раньше, чем накопители на магнитном диске. Бобинные накопители используются в суперЭВМ и mainframe. Ленточные накопители называются стримерами, они предназначены для создания резервных копий программ и документов, представляющих ценность. Запись может производиться на обычную видеокассету или на специальную кассету. **Емкость** такой кассеты до 1700 Мб, длина ленты 120 м, ширина 3.81 мм (2 - 4 дорожки). **Скорость считывания информации-до** 100 Кб/сек.

(2) **Диски** относятся к носителям информации с прямым доступом, т.е. ПК может обратиться к дорожке, на которой начинается участок с искомой информацией или куда нужно записать новую информацию, непосредственно.

Магнитные диски (МД)— в качестве запоминающей среды используются магнитные материалы со специальными свойствами, позволяющими фиксировать два направления намагниченности. Каждому из этих состояний ставятся в соответствие двоичные цифры — 0 и 1. Информация на МД записывается и считывается магнитными головками вдоль концентрических окружностей - **дорожек.** Каждая дорожка разбита на **сектора** (1 сектор = 512 б). Обмен между дисками и ОП происходит целым числом секторов. **Кластер** — минимальная единица размещения информации на диске, он может содержать один и более смежных секторов дорожки. При записи и чтении МД вращается вокруг своей оси, а механизм управления магнитной головкой подводит ее к выбранной для записи или чтения дорожке.

Данные на дисках хранятся в **файлах** — именованных областях внешней памяти, выделенных для хранения массива данных. Кластеры, выделяемые файлу, могут находиться в любом свободном месте дисковой памяти и необязательно являются смежными. Вся информация о том, где именно записаны кусочки файла, хранится в **таблице размещения файлов** FAT (file allocation table). Для пакетов МД (это диски, установленные на одной оси) и для двусторонних дисков вводится понятие **цилиндр** - совокупность дорожек МД, находящихся на одинаковом расстоянии от центра.

На ГМД магнитный слой наносится на гибкую основу. **Диаметр** ГМД: 5,25" и 3,5". **Емкость** ГМД от 180 Кб до 2,88 Мб. **Число дорожек** на одной поверхности - 80. **Скорость вращения** от 3000 до 7200 об/мин. Среднее **время доступа** 65 - 100 мс.

Каждая новая дискета перед работой должна быть **отформатирована,** т.е. создана структура записи информации на ее поверхности: разметка дорожек, секторов, записи

маркеров, таблицы FAT. Дискеты нужно хранить аккуратно, беречь от пыли, механических повреждений, воздействия магнитных полей, растворителей. Это основной недостаток этого вида накопителей.

НЖМД или «винчестеры» изготовлены из сплавов алюминия или из керамики и покрыты ферролаком, вместе с блоком магнитных головок помещены в герметически закрытый корпус. **Емкость** накопителей за счет чрезвычайно плотной записи достигает нескольких гигабайт, быстродействие также выше, чем у съемных дисков (за счет увеличения скорости вращения, т.к. диск жестко закреплен на оси вращения). Первая модель появилась на фирме IBM в 1973 г. Она имела емкость 16 Кб и 30 дорожек/30 секторов, что случайно совпало с калибром популярного ружья 30'730" «винчестер».

Диаметр ЖМД: 3,5" (есть 1,8" и 5,25"). **Скорость вращения** 7200 об/мин, **время доступа —** 6 мс.

Каждым ЖМД проходит процедуру **низкоуровневого форматирования** — на носитель записывается служебная информация, которая определяет разметку цилиндров диска на сектора и нумерует их, маркируются дефектные сектора для исключения их из процесса эксплуатации диска. В ПК имеется один или два накопителя. Один ЖД можно разбить при помощи специальной программы на несколько логических дисков и работать с ними как с разными ЖД.

Дисковые массивы RAID - применяются в машинах-серверах БД и в суперЭВМ, они представляют собой матрицу с резервируемыми независимыми дисками, несколько НЖМД объединены в один логический диск. Можно объединить до 48 физических дисков любой емкости, формирующих до 120 логических дисков (RAID7). **Емкость** таких дисков составляет до 5Т6 (терабайт=10^{12}).

НОД (накопители на оптических дисках) делятся на:

не перезаписываемые лазерно-оптические диски или компакт-диски (CD-ROM). Поставляются фирмой-изготовителем с уже записанной на них информацией. Запись на них возможна в лабораторных условиях лазерным лучом большой мощности. В оптическом дисководе ПК эта дорожка читается лазерным лучом меньшей мощности. Ввиду чрезвычайно плотной записи CD-ROM имеют емкость до 1,5 Гб, время доступа от 30 до 300 мс, скорость считывания данных от 150 до 1500 Кб/сек;

перезаписываемые CD-диски имеют возможность записывать информацию прямо с ПК, но для этого необходимо специальное устройство.

Магнитооптические диски (ZIP) — запись на такой диск производится под высокой температурой намагничиванием активного слоя, а считывание — лучом лазера. Эти диски

удобны для хранения информации, но оборудование стоит дорого. **Емкость** такого диска до 20,8 Мб, **время доступа** от 15 до 150 мс, **скорость считывания** информации до 2000 Кб/сек.

Контроллеры служат для обеспечения прямой связи с ОП, минуя МП, они используются для устройств быстрого обмена данными с ОП - НГМД, НЖД, дисплей и др., обеспечения работы в групповом или сетевом режиме. Клавиатура, дисплей, мышь являются медленными устройствами, поэтому они связаны с системной платой контроллерами и имеют в ОП свои отведенные участки памяти.

Порты бывают входными и выходными, универсальными (ввод - вывод), они служат для обеспечения обмена информацией ПК с внешними, не очень быстрыми устройствами. Информация, поступающая через порт, направляется в МП, а потом в ОП. Выделяют два вида портов:

последовательный — обеспечивает побитный обмен информацией, обычно к такому порту подключают модем;

параллельный — обеспечивает побайтный обмен информацией, к такому порту подключают принтер. Современные ПК обычно оборудованы 1 параллельным и 2 последовательными портами.

Видеомониторы — устройства, предназначенные для вывода информации от ПК пользователю. Мониторы бывают **монохромные** (зеленое или янтарное изображение, большая разрешающая способность) и **цветные.** Самые качественные RGB-мониторы, обладают высокой разрешающей способностью для графики и цвета. Используется тот же принцип электронной лучевой трубки как у телевизора. В портативных ПК используют **электролюминесцентные** или **жидкокристаллические** панели. Мониторы могут работать в текстовом и графическом режимах. В текстовом режиме изображение состоит из знакомест — специальных знаков, хранимых в видеопамяти дисплея, а в **графическом** изображение состоит из точек определенной яркости и цвета. Основные характеристики видеомониторов - разрешающая способность (от 600x350 до 1024x768 точек), число цветов (для цветных) -от 16 до 256, частота кадров фиксированная 60 Гц.

Принтеры — это устройства вывода данных из ЭВМ, преобразовывающие информационные ASCII-коды в соответствующие им графические символы и фиксирующие эти символы на бумаге. Принтеры - наиболее развитая группа внешних устройств, насчитывается более 1000 модификаций.

Принтеры бывают черно-белые или цветные по способу печати они делятся на:

матричные — в этих принтерах изображение формируется из точек ударным способом, игольчатая печатающая головка перемещается в горизонтальном направлении, каждая иголочка управляется электромагнитом и ударяет бумагу через красящую ленту. Количество

игл определяет качество печати (от 9 до 24), **скорость печати** 100-300 символов/сек, разрешающая способность 5 точек на мм;

струйные — в печатающей головке имеются вместо иголок тонкие трубочки - сопла, через которые на бумагу выбрасываются мельчайшие капельки чернил (12 - 64 сопла), **скорость печати** до 500 символов/сек, **разрешающая способность** - 20 точек на мм;

термографические — матричные принтеры, оснащенные вместо игольчатой печатающей головки головкой с термоматрицей, при печати используется специальная термобумага;

лазерные — используется электрографический способ формирования изображений, лазер служит для создания сверхтонкого светового луча, вычерчивающего на поверхности светочувствительного барабана контуры невидимого точечного электронного изображения. После проявления изображения порошком красителя (тонера), налипающего на разряженные участки, выполняется печать - перенос тонера на бумагу и закрепление изображения на бумаге при помощи высокой температуры. **Разрешение** у таких принтеров до 50 точек/мм, **скорость печати** - 1000 символов/сек.

Сканеры - устройства ввода в ЭВМ информации непосредственно с бумажного документа. Можно вводить тексты, схемы, рисунки, графики, фотографии и другую информацию. Файл, создаваемый сканером в памяти ЭВМ называется битовой картой. Существует два формата представления графической информации в ЭВМ:

растровый — изображение запоминается в виде мозаичного набора множества точек на экране монитора, редактировать такие изображения с помощью текстовых редакторов нельзя, эти изображения редактируют в Corel Draw, Adobe PhotoShop;

текстовый — информация идентифицируется характеристиками шрифтов, кодами символов, абзацев, стандартные текстовые процессоры предназначены для работы именно с таким представлением информации.

Битовая карта требует большого объема памяти, поэтому после сканирования битовые карты упаковывают с помощью специальных программ (PCX, GIF). Сканер подключается к параллельному порту. Сканеры бывают:

черно-белые и цветные **(число передаваемых цветов** от 256 до 65 536);

ручные перемещаются по изображению вручную, за один проход вводится небольшое количество информации (до 105 мм), **скорость считывания** - 5-50 мм/сек;

планшетные — сканирующая головка перемещается относительно оригинала автоматически, **скорость сканирования** -2-10 сек на страницу;

роликовые — оригинал автоматически перемещается относительно сканирующей головки;

проекционные - напоминают фотоувеличитель, внизу -сканируемый документ, сверху - сканирующая головка;

штрих-сканеры — устройства для считывания штрих-кодов на товарах в магазинах.

Разрешающая способность сканеров от 75 до 1600 точек/дюйм.

Манипуляторы - компьютерные устройства, управляемые руками оператора:

мышь — устройство для определения относительных координат (смещения относительно предыдущего положения или направления) движения руки оператора. Относительные координаты передаются в компьютер и при помощи специальной программы могут вызывать перемещения курсора на экране. Для отслеживания перемещения мыши используются различные виды датчиков. Самый распространенный - механический (шарик, к которому прикасаются несколько валиков), существует еще оптический датчик, обеспечивающий более высокую точность считывания координат;

джойстик — рычажный указатель - устройство для ввода направления движения руки оператора, их чаще используют для игр на компьютере;

дигитайзер или оцифровывающий планшет — устройство для точного ввода графической информации (чертежей, графиков, карт) в компьютер. Он состоит из плоской панели (планшета) и связанного с ней ручного устройства - пера. Оператор ведет вдоль графика перо, при этом абсолютные координаты поступают в компьютер.

Клавиатура — устройство для ввода информации в память компьютера. Внутри расположена микросхема, клавиатура связана с системной платой, нажатие любой клавиши продуцирует сигнал (код символа в системе ASCII -16-ричный порядковый номер символа в таблице), в памяти ЭВМ специальная программа по коду восстанавливает внешний вид нажатого символа и передает его изображение на монитор.

Конкретный набор компонент, входящих в данный компьютер, называется его **конфигурацией. Минимальная конфигурация ПК** необходимая для его работы включает в себя системный блок (там находятся МП, ОП, ПЗУ, НЖМД, НГМД), клавиатуру (как устройство ввода информации) и монитор (как устройство вывода информации).

Вопросы:

1. Какие бывают сканеры?
2. Какие существуют порты?

Практическая работа № 7

Тема: «Регистры процессора. Память.»

ЦЕЛЬ РАБОТЫ:

Приобретение навыков работы с регистрами процессора. и памятью.

2 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Прочитать задания к работе

2. Оформите отчет, который должен содержать:

–титульный лист (см. приложение);

–постановку задачи;

- описание пошагового исполнения;

– отчет о полученном результате

3. ЗАДАНИЯ К РАБОТЕ.

Для решения с помощью ЭВМ некоторой задачи должна быть разработана программа. Программа на языке ЭВМ представляет собой последовательность команд. Код каждой команды определяет выполняемую операцию, тип

адресации и адрес. Выполнение программы, записанной в памяти ЭВМ, осуществляется последовательно по командам в порядке возрастания адресов команд или в порядке, определяемом командами передачи управления.

Для того чтобы получить результат выполнения программы, пользователь должен:

- □ ввести программу в память ЭВМ;
- □ определить, если это необходимо, содержимое ячеек ОЗУ и РОН, содержащих исходные данные, а также регистров 1R и BR;
- □ установить в PC стартовый адрес программы;
- □ перевести модель в режим Работа.

Каждое из этих действий выполняется посредством интерфейса модели, описанного в главе 8. Ввод программы может осуществляться как в машинных кодах непосредственно в память модели, так и в мнемокодах в окно Текст программы с последующим ассемблированием.

Цель настоящей лабораторной работы — знакомство с интерфейсом модели ЭВМ, методами ввода и отладки программы, действиями основных классов команд и способов адресации. Для этого необходимо ввести в память ЭВМ и выполнить в режиме Шаг некоторую последовательность команд (определенную вариантом задания) и зафиксировать все изменения на уровне программно-доступных объектов ЭВМ, происходящие при выполнении этих команд.

Команды в память учебной ЭВМ вводятся в виде шестиразрядных десятичных чисел (см. форматы команд на рис. 8.3, коды команд и способов адресации в табл. 8.2—8.4).

В настоящей лабораторной работе будем программировать ЭВМ в машинных кодах.

9.1.2. Пример 1

Дана последовательность мнемокодов, которую необходимо преобразовать в машинные коды, занести в ОЗУ ЭВМ, выполнить в режиме Шаг и зафиксировать изменение состояний программно-доступных объектов ЭВМ (табл. 9.1).

Таблица 9.1. Команды и коды

Последовательность	Значения				
Команды	RD #20	WR 30	AD #5 D	WR @30	JNZ 002
Коды	21 1 020	22 0 030	23 1 005	22 2 030	12 0002

Введем полученные коды последовательно в ячейки ОЗУ, начиная с адреса 000. Выполняя команды в режиме Шаг, будем фиксировать изменения программно-доступных объектов (в данном случае это Асе, РС и ячейки ОЗУ 020 и 030) в табл. 9.2.

Таблица 9.2. Содержимое регистров

PC	Acc	M(30)	M(20)	PC	Acc	M(30)	M(20)
000	000000	000000	000000	004			000025
001	000020			002			
002		000020		003	000030		
003	000025			004			000030

9.1.3. Задание 1

1. Ознакомиться с архитектурой ЭВМ
2. Записать в ОЗУ "программу", состоящую из пяти команд— варианты задания выбрать из табл.. Команды разместить в последовательных ячейках памяти.
3. При необходимости установить начальное значение в устройство ввода IR.
4. Определить те программно-доступные объекты ЭВМ, которые будут изменяться при выполнении этих команд.
5. Выполнить в режиме Шаг введенную последовательность команд, фиксируя изменения значений объектов, определенных в п. 4, в таблице (см. форму табл. 9.2).
6. Если в программе образуется цикл, необходимо просмотреть не более двух повторений каждой команды, входящей в тело цикла.

Таблица 9.3. Варианты задания 1

№	IR	Команда 1	Команда 2	Команда 3	Команда 4	Команда S
1	000007	IN	MUL #2	WR10	WR @10	JNS 001
2	X	RD #17	SUB #9	WR16	WR @16	JNS 001
3	100029	IN	ADD #16	WR8	WR @8	JS 001
4	X	RD #2	MUL #6	WR 11	WR @11	JNZ 00
5	000016	IN	WR8	DIV #14	WR @8	JMP 002
6	X	RD #4	WR 11	RD @11	ADD #330	JS 000
7	000000	IN	WR9	RD @9	SUB#1	JS 001
8	X	RD 4	SUB #8	WR8	WR @8	JNZ 001
9	100005	IN	ADD #12	WR 10	WR @10	JS 004
10	X	RD 4	ADD #15	WR 13	WR @13	JMP 001
11	000315	IN	SUB #308	WR11	WR @11	JMP 001
12	X	RD #988	ADD #19	WR9	WR @9	JNZ 001
13	000017	IN	WR11	ADD 11	WR @11	JMP 002
14	X	RD #5	MUL #9	WR10	WR @10	JNZ 001

9.1.4. Содержание отчета

1. Формулировка варианта задания.
2. Машинные коды команд, соответствующих варианту задания.
3. Результаты выполнения последовательности команд в форме табл. 9.2.

9.1.5. Контрольные вопросы

1. Из каких основных частей состоит ЭВМ и какие из них представлены в модели?
2. Что такое система команд ЭВМ?
3. Какие классы команд представлены в модели?
4. Какие действия выполняют команды передачи управления?
5. Какие способы адресации использованы в модели ЭВМ? В чем отличие между ними?
6. Какие ограничения накладываются на способ представления данных модели ЭВМ?
7. Какие режимы работы предусмотрены в модели и в чем отличие между ними?
8. Как записать программу в машинных кодах в память модели ЭВМ.
9. Как просмотреть содержимое регистров процессора и изменить содермое некоторых регистров?
10. Как просмотреть и, при необходимости, отредактировать содержи ячейки памяти?

Практическая работа № 8-9.

Тема: Команды дисковой операционной системы, каталогов и файлов

1. Цель работы:

- изучение команд операционной системы MS-DOS;
- приобретение навыков работы с носителями информации.

2. Содержание работы:

- включение ПК, начальная загрузка;
- знакомство с основными устройствами ПК;
- изучение расположения клавиш на клавиатуре;
- выполнение команд операционной системы MS-DOS;

3. Порядок выполнения работы.

3.1. Знакомство с основными устройствами ПК.

- определите все основные устройства ПК: системный блок (процессор); дисплей (монитор);

клавиатура и печатающее устройство (принтер).

3.2. Изучение расположения клавиш на клавиатуре.

Найдите на клавиатуре основные группы клавиш:

- функциональные или программируемые клавиши (F1-F12), в каждой программе имеют свое назначение;
- основная алфавитно-цифровая группа клавиш служит для ввода информации. Расположение клавиш на ней аналогично расположению клавиш на обычной пишущей машинке;
- малая цифровая группа служит для ввода цифровой информации или для управления курсором. Для переключения режимов работы этой клавиатуры служит клавиша Num Lock. В режиме “включено” (лампочка Num Lock горит) работает режим ввода цифр, в режиме “выключено” (лампочка Num Lock не горит) клавиатура действует как панель управления курсором;
- группа клавиш для управления курсором: Home — перевести курсор на начало текущей строки; End — перевести курсор в конец текущей строки; Page Up — перевести курсор на страницу вверх; Page Down -перевести курсор на страницу вниз; Insert — переход между режимами вставки или замены; Delete — удаление символа в позиции которого стоит курсор; Backspace
- удаление символа слева от курсора;
- служебные клавиши: Esc — отменить текущее действие (набранную команду, предложенное действие и т.п.); Shift — смена регистра (верхний, нижний); CapsLock

- фиксация верхнего регистра; Tab — табулятор, действие его определяется конкретной программой; Ctr1 -используется в сочетании с другими клавишами, изменяя их назначение; Alt- действует аналогично клавише Ctr1; Enter — используется для выполнения указанной команды, в текстовых редакторах- переход на новую строку; Print Screen — печать содержимого экрана; Pause — временная приостановка выполнения каких-либо действий.

3.3. Включение ПК, начальная загрузка.

- Включите ПК тумблером, находящимся на передней панели или правой стенке системного блока;
- Затем включите дисплей тумблером на передней панели или на правой стенке; ПК начинает самотестирование и на экране дисплея появляются сменяющие друг друга сообщения. После успешного завершения самотестирования ПК выполняет попытку загрузки операционной системы с дискового устройства. Если попытка завершится неудачно (отсутствует дисковое устройство или дискета в устройстве), загрузка операционной системы происходит с жесткого диска (винчестера);
- По окончании загрузки на экране дисплея появляется приглашение операционной системы к вводу команд:

C:>_

Операционная система — это программа, которая автоматически загружается при включении компьютера. Персональный компьютер типа IBM PC работает под управлением операционной системы MS-DOS.

3.4. Выполнение команд операционной системы MS—DOS.

Операционная система имеет множество команд, которые делятся на внутренние и внешние. Внутренние команды входят в состав командного процессора Command.com. Внешние команды — это

самостоятельные программы, представляющие собой отдельные файлы.

3.4.1. Команда установки даты DATE и времени TIME.

(внутренние команды)

- введите команду DATE без параметров и получите текущую дату на вашем ПК;
- если дата неправильная, с помощью этой же команды установите новую дату;
- правильность ввода проверьте введением команды DATE без параметров.
- аналогичные действия проведите с командой TIME — проверка и установка времени.

3.4.2. Команда включения (отключения) верификации (проверки) обмена с дисковыми устройствами VERIFY.(внутренняя команда)

- включите верификацию командой VERIFY ;

- проверьте командой VERIFY;
- отключите верификацию командой VERIFY OFF;
- проверьте командой VERIFY.

3.4.3. Команда проверки версии MS—DOS VER. (внутренняя команда)

- проверьте версию MS-DOS командой VER.

3.4.4. Команда перехода на другой диск.

- задайте команду D: и проконтролируйте выполнение появлением приглашения системы

D:>_

- перейдите назад на диск C:
-

3.5. Команды работы с каталогами, (внутренние команды)

3.5.1. Команда создания каталога MD.

Создайте на рабочем винчестерском диске D: каталог, в котором в качестве имени используйте 8 символов своей фамилии:

а. перейти на рабочий винчестерский диск D:

б. введите команду

MD имя каталога

3.5.2. Команда смены каталога CD.

Перейдите в свой каталог, используя команду CD.

CD имя каталога

С помощью команды CD можно перейти в каталог на уровень выше:

CD..

и в корневой каталог:

CD\

В своем каталоге создайте подкаталог:

MD имя подкаталога

Примечание: Не забудьте имя подкаталога, также как и каталога образуется по правилам образования имени файла, т.е. до 8 символов, не должно включать стандартных зарезервированных имен устройств (CON:,PRN:, NUL, AUX, COM1, COM2, LPT1, LPT2, LPT3), может состоять из цифр (0-9), и знаков (“минус”, “подчеркивание”, расположенных в произвольном порядке, а также некоторых других символов: $, #, @, %, (,), {, }, ~, !)

— перейдите в свой подкаталог:

CD имя подкаталога

3.5.3. Команда удаления каталога RD.

Удаляется только пустой каталог.

3.6.Команды работы с файлами. (внутренние команды)

3.6.1. Команда копирования COPY.

Скопируйте с диска D из каталога WORK все файлы в свой подкаталог.

COPY D:\WORK\BABYTYPE*.* Имя диска:\имя каталога\ имя подкаталога

Скопируйте с диска D: из каталога BABYTYPE все файлы, имеющие расширение pdd на винчестерский рабочий диск в подкаталог со своим именем:

COPY D:\WORK\BABYTYPE *.pdd Имя диска:\имя каталога\имя подкаталога

ВНИМАНИЕ: В предыдущем пункте Вы скопировали в рабочий каталог все файлы, в том числе и файлы с именами, отвечающими шаблону *.pdd, поэтому при повторном копировании файлов с уже существующими именами, на экране дисплея появится сообщение, что данный файл уже существует (a1ready exists) и запрос: снять копирование (cancel) или переписать поверх существующего файла (overwrite).

Снимите копирование, оно не имеет смысла.

Создайте свой текстовый файл COPY CON имя файла

После ввода произвольного текста необходимо нажать комбинацию клавиш CTRL+Z, файл будет сохранен в вашем каталоге.

Прочитайте с экрана дисплея содержимое текстовых файлов 2 способами:

а. COPY имя вашего файла CON б. TYPE имя вашего файла

Используя команду COPY можно распечатать содержимое текстового файла на принтере:

COPY имя файла PRN

Создайте резервную копию своего файла , переименовав его в файл с именем имя вашего файла.01 COPY имя вашего файла имя вашего файла.01

Используя команду COPY можно объединять несколько файлов в один:

COPY имя файла1+имя файла2+… имя файла. Если не указывать имя файла, полученного в результате объединения, объединенный файл получит имя первого файла.

3.6.2. Команда переименования файлов REN

Переименуйте свой файл в вашем каталоге в файл имя файла.98:

REN имя файла имя файла.98

3.6.3. Команда чтения оглавления каталога DIR.

Команда DIR позволяет просмотреть содержимое любого каталога (корневого, текущего, указанного конкретно). При использовании команды DIR можно указывать шаблон имен файлов. Ключи команды DIR:

ключ /P — позволяет выводить информацию постранично, с остановками после заполнения экрана монитора. Информация выводится в полном виде (имена файлов, их объем в байтах, дата и время создания файлов);

ключ /W — информация выводится в сжатом виде (только имена файлов и каталогов), в строчку.

Выполните описанные ниже действия и объясните смысл:

а. C>DIR D:\WORK\KURSDOS

б. C>DIR D:\WORK\KURSDOS\VOPROSI.*

в. C> DIR D:\WORK\KURSDOS*.d

г. C>DIR

д. C>DIR /P

е. C>DIR /W

3.6.4. Команды удаления файлов ERASE, DEL

Удалите 2 способами файл и группу файлов в своем каталоге:

а. ERASE имя файла б. DEL *.pdd

Во втором случае на экране дисплея появится запрос:

Are you sure? (y/n) Вы уверены? (да/нет) Ответьте Y (yes) — да.

После удаления всех файлов удалите свой подкаталог и свой каталог, для этого:

- выйдите из своего подкаталога: CD..
- удалите свой подкаталог:

RD имя подкаталога

- выйдите из своего каталога: CD\
- удалите свой каталог:

RD имя каталога

3.7. Команды работы с дисками

(внешние команды)

3.7.1. Команда форматирования диска —FORMAT.

- введите команду форматирования дискеты, находящейся в дисковом устройстве A: и подождите проверки правильности введенной команды преподавателем:

C:> FORMAT A: /S /T:40/N:9

- Ключ /S — на форматируемый диск записываются Системные файлы MS-DOS (IO.SYS, MSDOS.SYS, COMMAND.COM);
- /T:40 — дискета размечается на 40 дорожек,
- /N:9 — девять секторов на каждой дорожке.

При таком задании ключей объем дискеты равен 360 Кб.

При форматировании дискеты команда FORMAT выводит сообщение

Insert diskette for drive A: and strike ENTER when ready

(Вставьте дискету в дисковод A: и нажмите клавишу ENTER)

По окончании форматирования выводится сообщение о полном объеме дискеты; объеме памяти, занятой DOS; объеме памяти, доступной для записи; объеме памяти, занятой сбойными участками.

3.7.2. Копирование диска на диск — DISKCOPY.

DISKCOPY позволяет получать точную копию диска. Для этого нужно иметь диск источник (SOURSE) и диск приемник (TARGET).

- если дисковое устройство одно, то копирование проводится с диска A: на диск A:
- если дисковых устройств два, то копирование производится с диска A: на диск B:

Программа DISKCOPY копирует дискеты по дорожкам.

Если форматы дискет разные, то программа выдаст сообщение:

Drive types or diskette types not compatible

(Типы дисководов или дискет несовместимы)

или Disks must of the same size

(Дискеты должны быть одного размера)

Если дискета не форматирована, то программа ее форматирует. Формат тот же что и у исходной дискеты.

- введите команду

C:> DISKCOPY A: A:

- после считывания исходной дискеты следует указание вставить дискету приемник.

3.7.3. Проверка точности копирования дискет —DISKCOMP.

- перейдите на диск C:
- сравните две дискеты командой DISKCOMP A: A:. Сравнение заканчивается вопросом: Сравнивать еще? (да/нет). Ответьте нет.

3.7.4. Проверка дисков и файлов — CHKDSK.

- введите команду

C:> CHKDSK A: /F

- проверяется диск A:. Ключ /F задает режим, при котором системой будут предприниматься попытки автоматического устранения обнаруженных ошибок (указывается только при проверке диска).
- введите команду

C:> CHKDSK A:*.*

- команда проверяет все файлы на диске A: .

В сообщении указывается емкость диска, количество скрытых файлов (если они есть), объем памяти, который они занимают, число пользовательских файлов и их объем, свободный объем памяти на диске, дефектные дорожки или сектора и их объем, общий объем памяти ЭВМ и свободная память после размещения DOS.

- выполнить команду для диска D:, но без ключа.

Итог

Итак, Вы познакомились с составными частями ПЭВМ IBM PC/AT и научились работать с командами операционной системы MS-DOS: форматировать и проверять дискеты, копировать дискету на дискету, читать и выводить на печать оглавление активного или любого другого диска, каталога, подкаталога, создавать, по мере необходимости, собственные каталоги (подкаталоги), копировать, переименовывать в них файлы или группы файлов по маске, приобрели навыки удаления ненужной (устаревшей) информации; закрепили приемы работы с активным дисководом, переходом от одного накопителя к другому.

Задание повышенного уровня сложности (дополнительное).

1. Проанализируйте истины или ложны приведенные ниже высказывания:

а. команда DIR не меняет содержимого дисков, каталогов, подкаталогов, а лишь показывает их оглавление.

б. команда DIR в своих разновидностях изменяет содержимое каталогов, в зависимости от шаблона (умолчаний — “*”или комбинация символов “?”).

2. Возможно использование сокращенных обозначений в маршрутах для указания текущего каталога, расположенного на один уровень выше. Например:

а. DIR.. — выдаются файлы в каталоге на уровень выше текущего.

б. DIR../.. — на экран дисплея выводится оглавление всех файлов в каталоге на 1 уровень выше.

Приведите примеры содержимого винчестерских дисков C:, D:, E:.

3. Прочитать с экрана дисплея содержимое текстовых файлов inf.txt и fiz.txt с разбиением на порции: командой MORE.

Контрольные вопросы к защите лабораторной работы:

1. Понятие и назначение ДОС.
2. Что такое файл? Как задается имя файла?
3. Что такое каталог (директория), подкаталог?
4. Полный путь к файлу?
5. Что такое шаблон (маска) имен файлов?

6. Какими символами обозначаются дисководы?

7. Понятие внешние и внутренние команды ОС.

8. Что означает операция форматирования и как ее выполнить?

9. Как загрузить ОС с дискеты, а не с жесткого диска? Опция для указания записи системных файлов при форматировании.

10.Как переписать информацию с одной дискеты на другую? Требования, предъявляемые к дискетам при этом?

11.Как проверить правильность копирования дискет?

12.Как проверить дискету на наличие сбойных участков?

13.Команды работы с каталогами.

14.Команды работы с файлами.

15.Команды общесистемного назначения

Практическая работа № 10

Тема: Стек. Подпрограммы. Циклы

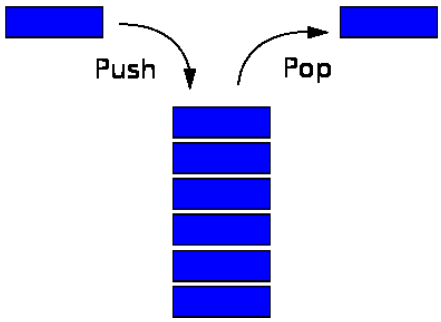
***Цель работы*:** ознакомление с организацией стека микропроцессоров Intel x86, организацией и видами подпрограмм и циклов; получение навыков использования стековых операций, использования подпрограмм и циклов языка Ассемблер при разработке программ.

Краткие теоретические сведения

1. Стек

1.1. Стек – как структура данных

Стек (англ. stack — стопка) — структура данных с методом доступа к элементам LIFO (англ. Last In — First Out, «последним пришёл — первым вышел»). Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно снять верхнюю.



Добавление элемента, называемое также проталкиванием (push), возможно только в вершину стека (добавленный элемент становится первым сверху). Удаление элемента, называемое также выталкивание (pop), возможно также только из вершины стека, при этом, второй сверху элемент становится верхним.

Стеки широко применяются в вычислительной технике — в частности, для отслеживания точек возврата из подпрограмм используется стек вызовов, который является неотъемлемой частью архитектуры большинства современных процессоров. Языки программирования высокого уровня также используют стек вызовов для передачи параметров при вызове процедур.

Арифметические сопроцессоры, программируемые микрокалькуляторы и язык Forth используют стековую модель вычислений.

В вычислительной технике стек часто сравнивается с магазином – память «магазинного типа» — по аналогии с магазином в огнестрельном оружии (стрельба начнётся с патрона, заряженного последним)

1.2. Стек вызовов

Стек вызовов (от англ. *call stack*; применительно к процессорам — просто «**стек**») — в теории вычислительных систем, LIFO-стек, хранящий информацию для возврата управления из подпрограмм (процедур) в программу (или подпрограмму, при вложенных или рекурсивных вызовах) и/или для возврата в программу из обработчика прерывания (в том числе при переключении задач в многозадачной среде).

При вызове подпрограммы или возникновении прерывания, в стек заносится *адрес возврата* — адрес в памяти следующей инструкции приостановленной программы и управление передается подпрограмме или подпрограмме-обработчику. При последующем вложенном или рекурсивном вызове, прерывании подпрограммы или обработчика прерывания, в стек заносится очередной адрес возврата и т. д.

При возврате из подпрограммы или обработчика прерывания, адрес возврата снимается со стека и управление передается на следующую инструкцию приостановленной (под-)программы.

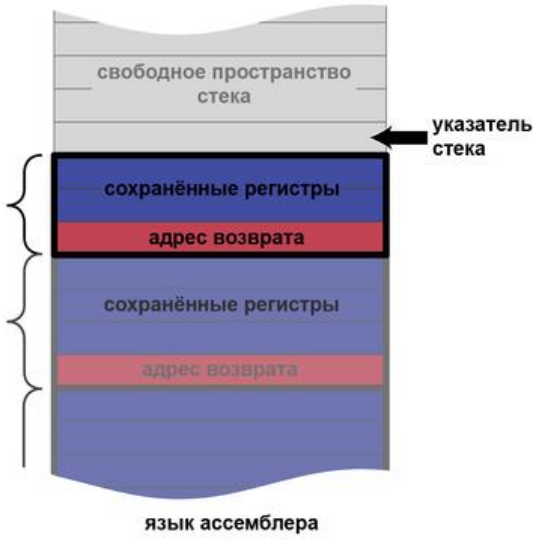
Реализация

Стек вызовов может быть реализован как в виде специализированного стекового регистра ограниченной глубины, так и в виде указателя вершины стека в оперативную память или регистровый файл процессора.

При отсутствии или ограниченности стека, вложенные вызовы исключены или их количество ограничено. При необходимости бóльшей вложенности, стек вызовов или его расширение могут быть реализовано программно.

Поддержка

Вызов подпрограммы и возвраты из подпрограмм и обработчиков прерываний, как правило, выполняются специализированными инструкциями процессора. Кроме инструкций вызовов и возвратов, процессоры часто имеют инструкции для использования стека вызовов также и под сохранение данных — их помещения в стек, снятия со стека, модификации содержимого стека.



Инструкции вызова, возврата и работы со стеком могут отличаться по размеру сохраняемых данных (в этом случае необходимо использовать соответствующие друг другу инструкции или их эквиваленты).

Иногда процедуры возврата из подпрограммы и обработчика прерываний отличаются друг от друга, и также требуют разных команд (например, при возврате из прерывания часто необходимо восстановить из стека регистр флагов и/или разрешить обработку конкурентных прерываний, которая может автоматически запрещаться при вызове обработчика).

Использование

Стек вызовов может использоваться для различных нужд, но основное его назначение — отслеживать место, куда каждая из вызванных процедур должна вернуть

управление после своего завершения. Для этого при вызове процедуры (командами вызова) в стек заносится адрес команды, следующей за командой вызова («адрес возврата»). По завершении вызванная процедура должна выполнить команду возврата для перехода по адресу из стека.

Кроме адресов возврата в стеке могут сохраняться другие данные, например:

- значения регистров с их последующим восстановлением
- другие произвольные данные

В многозадачных системах, каждая задача как правило имеет свой собственный стек, и при переключении задачи указатель стека процессора переставляется на него.

Стек может быть использован нестандартно, например:

- вызванная подпрограмма может модифицировать адрес возврата, например, при выборке аргументов расположенных *следом за инструкцией вызова*
- возврат не в вызывающую подпрограмму, а в предыдущую по стеку (при обработке нештатной ситуации)

1.3. Стековые команды

Таблица 5.1 - Стековые команды

НАЗВАНИЕ	***МНЕМОНИКА И ФОРМАТ***	***ОПИСАНИЕ***
Включить в стек	PUSH SRC	(SP) ← (SP)-2 ((SP)+1:(SP) ← (SRC)
Извлечь из стека	POP DST	(DST) ← ((SP)+1:(SP) (SP) ← (SP)+2

2. Подпрограммы

Подпрограмма (англ. *subroutine*) — поименованная или иным образом идентифицированная часть компьютерной программы, содержащая описание определённого набора действий. Подпрограмма может быть многократно *вызвана* из разных частей программы.

2.1. Назначение подпрограмм.

Подпрограммы изначально появились как средство оптимизации программ по объёму занимаемой памяти — они позволили не повторять в программе идентичные блоки кода, а описывать их однократно и вызывать по мере необходимости. К настоящему времени данная функция подпрограмм стала вспомогательной, главное их назначение — структуризация программы с целью удобства её понимания и сопровождения.

- Выделение набора действий в подпрограмму и вызов её по мере необходимости позволяет логически выделить целостную подзадачу, имеющую типовое

решение. Такое действие имеет ещё одно (помимо экономии памяти) преимущество перед повторением однотипных действий: любое изменение (исправление ошибки, оптимизация, расширение функциональности), сделанное в подпрограмме, автоматически отражается на всех её вызовах, в то время как при дублировании каждое изменение необходимо вносить в каждое вхождение изменяемого кода.

- Даже в тех случаях, когда в подпрограмму выделяется однократно производимый набор действий, это оправдано, так как позволяет сократить размеры целостных блоков кода, составляющих программу, то есть сделать программу более понятной и обозримой...

2.2. Механизм подпрограмм, их описание и вызов

В случае ассемблера подпрограмма представляет собой последовательность команд (операторов), отдельную от основной части программы и имеющую в конце специальную команду выхода из подпрограммы. Обычно подпрограмма также имеет имя, по которому её можно вызвать, хотя ряд языков программирования допускает использование и неименованных подпрограмм.

Вызов подпрограммы выполняется с помощью команды вызова, включающей в себя имя подпрограммы.

Внимание! Ни в коем случае нельзя попадать в подпрограмму любым способом кроме команды вызова подпрограммы CALL! В противном случае команда возврата из подпрограммы передаст управление случайному адресу! По этому адресу могут быть расположены данные, которые в этом случае будут интерпретированы как программа.

Очень часто требуется из одной подпрограммы обращаться к другой подпрограмме. Такое обращение к подпрограмме называется вложенным. Количество вложенных подпрограмм называется уровнем вложенности подпрограмм. Максимально допустимый уровень вложенности подпрограмм определяется количеством ячеек памяти, предназначенных для хранения адресов возврата из подпрограмм, т.е. размером сегмента стека.

2.3. Параметры подпрограмм

Подпрограммы часто используются для многократного выполнения стереотипных действий над различными данными. Подпрограмма обычно имеет доступ к объектам данных, описанным в основной программе, поэтому для того, чтобы передать в подпрограмму обрабатываемые данные, их достаточно присвоить, например, глобальным переменным.

3. Циклы

Цикл — разновидность управляющей конструкции, предназначенная для организации многократного исполнения набора инструкций. Также циклом может называться любая

многократно исполняемая последовательность инструкций, организованная любым способом (например, с помощью условного перехода).

Последовательность инструкций, предназначенная для многократного исполнения, называется *телом цикла*. Единичное выполнение тела цикла называется итерацией. Выражение определяющее, будет в очередной раз выполняться итерация, или цикл завершится, называется *условием выхода* или *условием окончания цикла* (либо *условием продолжения* в зависимости от того, как интерпретируется его истинность — как признак необходимости завершения или продолжения цикла). Переменная, хранящая текущий номер итерации, называется *счётчиком итераций* цикла или просто *счётчиком цикла*. Цикл не обязательно содержит счётчик, счётчик не обязан быть один — условие выхода из цикла может зависеть от нескольких изменяемых в цикле переменных, а может определяться внешними условиями (например, наступлением определённого времени), в последнем случае счётчик может вообще не понадобиться.

Исполнение любого цикла включает первоначальную инициализацию переменных цикла, проверку условия выхода, исполнение тела цикла и обновление переменной цикла на каждой итерации.

3.1. Виды циклов

Безусловные циклы

Иногда в программах используются циклы, выход из которых не предусмотрен логикой программы. Такие циклы называются безусловными, или бесконечными. Специальных синтаксических средств для создания бесконечных циклов, ввиду их нетипичности, языки программирования не предусматривают, поэтому такие циклы создаются с помощью конструкций, предназначенных для создания обычных (или *условных*) циклов – в ассемблере – с помощью команд безусловных переходов.

Цикл с предусловием, цикл с постусловием

Цикл с предусловием — цикл, который выполняется пока истинно некоторое условие, указанное перед его началом. Это условие проверяется **до** выполнения тела цикла, поэтому тело может быть не выполнено ни разу (если условие с самого начала ложно).

Цикл с постусловием — цикл, в котором условие проверяется **после** выполнения тела цикла. Отсюда следует, что тело **всегда выполняется** хотя бы один раз.

На языке ассемблера циклы с предусловием или с постусловием реализуются с помощью команд условных переходов.

Цикл с выходом из середины

Синтаксически такой цикл оформляется с помощью трёх конструкций: начала цикла, конца цикла и команды выхода из цикла. Конструкция начала маркирует точку программы, в которой начинается тело цикла, конструкция конца — точку, где тело заканчивается. Внутри

тела должна присутствовать команда выхода из цикла, при выполнении которой цикл заканчивается и управление передаётся на оператор, следующий за конструкцией конца цикла. Естественно, чтобы цикл выполнился более одного раза, команда выхода должна вызываться не безусловно, а только при выполнении условия выхода из цикла.

Принципиальным отличием такого вида цикла от рассмотренных выше является то, что часть тела цикла, расположенная после начала цикла и до команды выхода, выполняется всегда (даже если условие выхода из цикла истинно при первой итерации), а часть тела цикла, находящаяся после команды выхода, не выполняется при последней итерации.

Легко видеть, что с помощью цикла с выходом из середины можно легко смоделировать и цикл с предусловием (разместив команду выхода в начале тела цикла), и цикл с постусловием (разместив команду выхода в конце тела цикла).

Цикл со счётчиком

Цикл со счётчиком — цикл, в котором некоторая переменная изменяет своё значение от заданного начального значения до конечного значения с некоторым шагом, и для каждого значения этой переменной тело цикла выполняется один раз. В языке программирования Ассемблер для МП архитектуры x86 цикл со счетчиком реализуется командами LOOP (таблица 5.2). В качестве счётчика (так называемой «переменной цикла»), используется регистр CX. Перед циклом в нем задается требуемое количество проходов. Каждое выполнение команды LOOP декрементирует содержимое CX, и выходит из цикла, когда остаточное количество проходов равно нулю, иначе переходит на метку, символизирующую начало цикла.

```
    mov cx,<количество проходов цикла>
<метка>:            ; метка начала тела цикла
    ……              ; тело цикла
    loop <метка>    ; команда зацикливания
```

Цикл со счётчиком всегда можно записать как условный цикл, перед началом которого счётчику присваивается начальное значение, а условием выхода является достижение счётчиком конечного значения; к телу цикла при этом добавляется оператор изменения счётчика на заданный шаг. Однако специальные операторы цикла со счётчиком могут эффективнее транслироваться, так как формализованный вид такого цикла позволяет использовать специальные процессорные команды организации циклов.

3.2. Команды циклов

Таблица 5.2 - Команды циклов

НАЗВАНИЕ	***МНЕМОНИКА И ФОРМАТ***	***ПРОВЕРЯЕМОЕ УСЛОВИЕ***
Зациклить	LOOP OPR	(CX) ≠ 0
Зациклить, пока нуль или	LOOPZ OPR	ZF=1 и (CX) ≠ 0

равно		
Зациклить, пока не нуль или не равно	LOOPNZ OPR	ZF=0 и (CX) ≠ 0
Перейти по CX	JCXZ OPR	(CX) = 0

За исключением команды JCXZ, которая не изменяет (CX), производится (CX)←(CX)-1, затем, если проверяемое условие удовлетворяется, (IP)←(IP)+D8 с расширением знака; в противном случае (IP) не изменяется и программа продолжается в естественном порядке.

Не модифицируются все флажки условий.

Примеры решения задач:

```
;структура программы, использующей подпрограммы:
s_s      segment stack "stack"
           dw 12 dup(?)
s_s      ends
d_s      segment
aa        dw 10
d_s      ends
c_s      segment
           assume ss:s_s,ds:d_s,cs:c_s;
begin:     mov ax,d_s
           mov ds,ax;
           call pr1     ;вызов подпрограммы
          ..............
           call pr1     ;вызов подпрограммы
          ..............
           mov ah,4ch
           int 21h

pr1      proc near        ;начало подпрограммы (ближний вызов)
          ............

          ret            ;команда возврата  на следующую команду
после
                          ;вызова прцедуры
pr1     endp              ;конец подпрограммы
c_s     ends
          end begin
```

Задание на практическую работу:

1. Реализация программы, использующей стековые команды:

1. Узнать у преподавателя вариант работы.
2. Составить программу согласно условию и варианту:

   Входные числа заданы в сегменте данных в переменной X. При помощи стековых операций изменить порядок следования чисел.

3. Транслировать ассемблерный код. В случае нахождения ошибок – устранить их и повторить трансляцию.
4. Скомпоновать программу редактором связей.
5. Запустить программу в отладчике, в случае обнаружения ошибок в работе программы исправить их, повторить п.п.3, 4
6. Показать работу программы преподавателю

Таблица 5.3 - данные для реализации согласно варианту

Вариант	**Входные числа**	**Выходной порядок чисел**
1	1h, 2h, 3h, 4h, 5h, 6h, 7h	5h, 4h, 1h, 6h, 7h, 2h, 3h
2	5h, 33h, 90h, 10h, 4h, 34h	33h, 10h, 4h, 5h, 34h, 90h
3	11h, 33h, 1h, 12h, 9h, 12h	12h, 9h, 11h, 1h, 12h, 33h
4	10h, 2h, 44h, 66h, 77h, 12h, 1h	77h, 66h, 1h, 2h, 44h, 12h, 10h
5	9h, 22h, 53h, 6h, 2h, 8h, 3h	3h, 22h, 9h, 6h, 2h, 8h, 53h,
6	9h, 8h, 7h, 6h, 5h, 4h, 3h	7h, 9h, 4h, 3h, 8h, 6h, 5h
7	1h, 3h, 5h, 7h, 9h, 2h, 4h	7h, 9h, 5h, 2h, 1h, 3h, 4h
8	44h, 21h, 42h, 90h, 22h, 10h, 1h	42h, 10h, 1h, 90h, 44h, 21h, 22h
9	88h, 22h, 1h, 2h, 4h, 5h ,6h	22h, 1h, 4h, 5h ,6h, 88h, 2h
10	89h, 20h, 33h, 5h, 2h, 51h, 25h	2h, 51h, 5h, 25h, 89h, 20h, 33h

2. Реализация программы, использующей подпрограммы:

1. Составить программу согласно условию:

   Оформить в виде подпрограммы и вывести на экран сообщение «Hello World!».
2. Транслировать ассемблерный код. В случае нахождения ошибок – устранить их и повторить трансляцию.
3. Скомпоновать программу редактором связей.
4. Запустить программу в консольном режиме, в случае обнаружения ошибок в работе программы исправить их, повторить п.п.2,3
5. Показать работу программы преподавателю

3. Реализация программы, использующей подпрограммы:

1. Составить программу согласно условию:

   Оформить в виде подпрограммы и вывести на экран 7 сообщений «Hello World!».
2. Транслировать ассемблерный код. В случае нахождения ошибок – устранить их и повторить трансляцию.
3. Скомпоновать программу редактором связей.

4. Запустить программу в консольном режиме, в случае обнаружения ошибок в работе программы исправить их, повторить п.п.2,3
5. Показать работу программы преподавателю

4. Реализация программы, использующей подпрограммы:

1. Составить программу согласно условию:
   Оформить в виде подпрограмм 2 сообщения: «Hello!» и «World!». Вывести их на экран в виде:
   Hello!
   Hello!
   Hello!
   World!
   Hello!
   World!
2. Транслировать ассемблерный код. В случае нахождения ошибок – устранить их и повторить трансляцию.
3. Скомпоновать программу редактором связей.
4. Запустить программу в консольном режиме, в случае обнаружения ошибок в работе программы исправить их, повторить п.п.2,3
5. Показать работу программы преподавателю

5. Реализация программы, использующей циклы:

1. Составить программу согласно условию:
   Оформить в виде подпрограммы и вывести на экран 50 сообщений «Hello World!». Использовать операции циклов.
2. Транслировать ассемблерный код. В случае нахождения ошибок – устранить их и повторить трансляцию.
3. Скомпоновать программу редактором связей.
4. Запустить программу в консольном режиме, в случае обнаружения ошибок в работе программы исправить их, повторить п.п.2,3
5. Показать работу программы преподавателю

Содержание отчета по практической работе:

1. Титульный лист
2. Цель работы
3. Краткие теоретические сведения
4. Описание реализованных программ
5. Исходный ассемблерный код реализованных программ

6. Входные и выходные данные по программе.
7. Выводы по практической работе.

Практическая работа № 13

Тема: «Принципы работы КЭШ памяти»

ЦЕЛЬ РАБОТЫ

Приобретение навыков работы с кэш-памятью.

Проверить работу различных алгоритмов замещения при различных режимах записи в кэш-память.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

- Прочитать методические указания
- Разобрать приведенные примеры
- Выполнить задания к работе
- Ответить на контрольные вопросы

Ввести в модель учебной ЭВМ текст своего варианта программы (см. табл. 2), ассемблировать его и сохранить на диске в виде txt-файла.

Установить параметры кэш-памяти размером 4 ячейки, выбрать режим записи и алгоритм замещения в соответствии с первой строкой своего варианта из табл. 1.

В шаговом режиме выполнить программу, фиксируя после каждого шага состояние кэш-памяти.

Для одной из команд записи (WR) перейти в режим Такт и отметить, в каких микрокомандах происходит изменение кэш-памяти.

Для кэш-памяти размером 8 ячеек установить параметры в соответствии со второй строкой своего варианта из табл. 1 и выполнить программу в шаговом режиме еще раз, фиксируя последовательность номеров замещаемых ячеек кэш-памяти.

Оформите отчет, который должен содержать:

- титульный лист (см. приложение);

постановку задачи;

- Вариант задания — текст программы и режимы кэш-памяти.
- Последовательность состояний кэш-памяти размером 4 ячейки при однократном выполнении программы (команды 1—7).
- Последовательность микрокоманд при выполнении команды WR С отметкой тех микрокоманд, в которых возможна модификация кэш-памяти.
- Для варианта кэш-памяти размером 8 ячеек — последовательность номеров замещаемых ячеек кэш-памяти для второго варианта параметров кэш памяти при двукратном выполнении программы (команды 1—7).
- Вывод

В качестве задания предлагается некоторая короткая "программа" (табл. 9.14), которую

необходимо выполнить с подключенной кэш-памятью (размером 4 и 8 ячеек) в шаговом режиме для следующих двух вариантов алгоритмов замещения (табл.).

Таблица 1. Пояснения к вариантам задания

Номера вариантов	Режим записи	Алгоритм замещения
1,7, 11	Сквозная	СЗ, без учета бита записи
	Обратная	О, с учетом бита записи
2,5,9	Сквозная	БИ, без учета бита записи
	Обратная	О, с учетом бита записи
3,6, 12	Сквозная	О, без учета бита записи
	Обратная	СЗ, с учетом бита записи
4, 8, 10	Сквозная	БИ, без учета бита записи
	Обратная	БИ, с учетом бита записи

Таблица 2. Варианты задания 7

№ вари-	Номера команд программы						
	1	2	3	4	5	6	7
1	RD #12	WR 10	WR §10	ADD 12	WR R0	SUB 10	PUSH R0
2	RD #65	WRR2	MOV R4,R2	WR 14	PUSH R2	POP R3	CALL 002
3	RD #16	SUB #5	WR 9	WR @9	WR R3	PUSH R3	POP R4
4	RD #99	WR R6	MOV R7,R6	ADD R7	PUSH R7	CALL 006	POP R8
5	RD #11	WR R2	WR -@R2	PUSH R2	CALL 005	POP R3	RET
6	RD #19	SUB #10	WR9	ADD #3	WR ©9	CALL 006	POPR4
7	RD #6	CALL 006	WR11	WRR2	PUSH R2	RET	JMP 002
8	RD#8	WRR2	WR @R2+	PUSH R2	POP R3	WR -@R3	CALL 003
9	RD #13	WR14	WR@14	WR@13	ADD 13	CALL 006	RET
10	RD #42	SUB #54	WR16	WR@16	WRR1	ADD	PUSH Rl
11	RD #10	WRR5	ADD R5	WRR6	CALL 005	PUSH R6	RET
12	JMP 006	RD #76	WR 14	WRR2	PUSH R2	RET	CALL 001

Не следует рассматривать заданную последовательность команд как фрагмент программы[1]. Некоторые конструкции, например, последовательность команд PUSH R6, RET в общем случае не возвращает программу в точку вызова подпрограммы. Такие группы команд введены в задание для того, чтобы обратить внимание студентов на особенности функционирования стека.

Напомним, что программа определяется как последовательность команд, выполнение которых позволит получить некий результат.

9.7.4. Контрольные вопросы

В чем смысл включения кэш-памяти в состав ЭВМ?

Как работает кэш-память в режиме обратной записи? Сквозной записи?

Как зависит эффективность работы ЭВМ от размера кэш-памяти?

В какую ячейку кэш-памяти будет помещаться очередное слово, если свободные ячейки отсутствуют?

Какие алгоритмы замещения ячеек кэш-памяти вам известны?

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ

ПРОГРАММНАЯ МОДЕЛЬ КЭШ-ПАМЯТИ

К описанной в разд. 8.1 программной модели учебной ЭВМ может быть подключена программная модель кэш-памяти, структура которой в общем вид* отображена на рис. 5.2. Конкретная реализация кэш-памяти в описываемой программной модели показана на рис. 8.15.

Кэш-память содержит N ячеек (в модели N может выбираться из множества {4, 8, 16, 32}), каждая из которых включает трехразрядное поле тега (адреса ОЗУ), шестиразрядное поле данных и три однобитовых признака (флага):

- Z — признак занятости ячейки;
- U — признак использования;
- W — признак записи в ячейку.

Таким образом, каждая ячейка кэш-памяти может дублировать одну любую ячейку ОЗУ, причем отмечается ее занятость (в начале работы модели щ ячейки кэш-памяти свободны, VZ, = 0), факт записи информации в ячейк\ t$ время пребывания ее в кэш-памяти, а также использование ячейки (т. с бое обращение к ней).

Текущее состояние кэш-памяти отображается на экране в отдельном окне в форме таблицы, причем количество строк соответствует выбранному числу ячеек кэш. Столбцы таблицы определяют содержимое полей ячеек, например, так, как показано в табл. 8.3.

	Теги	Данные	Z	и	W
	012	220152	1	0	0
	013	211003	1	1	0
	050	000025	1	1	1
	000	000000	0	0	0

Для настройки параметров кэш-памяти можно воспользоваться диалоговым окном Кэш-память, вызываемым командой Вид | Кэш-память, а затем нажать первую кнопку на панели инструментов открытого окна. После этих действий появится диалоговое окно Параметры кэш-памяти, позволяющее выбрать размер кэш-памяти, способ записи в нее информации и алгоритм замещения ячеек.

Напомним, что при сквозной записи при кэш-попадании в процессорных циклах записи осуществляется запись как в ячейку кэш-памяти, так и в ячейку ОЗУ, а при обратной записи — только в ячейку кэш-памяти, причем эта ячейка отмечается битом записи ($W_t := 1$). При очистке ячеек, отмеченных битом записи, необходимо переписать измененное значение пони пни

соответствующую ячейку ОЗУ.

При кэш-промахе следует поместить в кэш-память адресуемую npoiii i ■ ячейку. При наличии свободных ячеек кэш-памяти требуемое слон») i ется в одну из них (в порядке очереди). При отсутствии свободных >г дует отыскать ячейку кэш-памяти, содержимое которой можно удалим. сав на его место требуемые данные (команду). Поиск такой ячейки ООН ляется с использованием алгоритма замещения строк.

В модели реализованы три различных алгоритма замещения строк:

случайное замещение, при реализации которого номер ячейки к >ш I выбирается случайным образом;

очередь, при которой выбор замещаемой ячейки определяется врем пребывания ее в кэш-памяти;

бит использования, случайный выбор осуществляется только in i которые имеют нулевое значение флага использования.

Напомним, что бит использования устанавливается в 1 при любом обра к ячейке, однако, как только все биты £/, установятся в 1, все они сбрасываются в 0, так что в кэш всегда ячейки разбиты на два не пер! щихся подмножества по значению бита U— те, обращение к которы стоялось относительно недавно {после последнего сброса вектора II) и значение U= 1, иные— со значением U = 0 являются "кандидатами i ление" при использовании алгоритма замещения "бит использовании". Если в параметрах кэш-памяти установлен флаг "с учетом бита запио\ все три алгоритма замещения осуществляют поиск "кандидата на прежде всего среди тех ячеек, признак записи которых не установк отсутствии таких ячеек (что крайне маловероятно) — среди всех я памяти. При снятом флаге "с учетом бита записи" поиск осуществлЯ! всем ячейкам кэш-памяти без учета значения W.

Оценка эффективности работы системы с кэш-памятью определяемсн ЧШ I кэш-попаданий по отношению к общему числу обращений к памяти \ вая разницу в алгоритмах записи в режимах сквозной и обратной запж фективность использования кэш-памяти вычисляется по следующим | жениям (соответственно для сквозной и обратной записи):

Тема: АЛГОРИТМЫ ЗАМЕЩЕНИЯ СТРОК КЭШ-ПАМЯТИ

Цель: изучение влияния параметров кэш-памяти и выбранного алгоритма замещения на эффективность работы системы. Эффективность в данном случае оценивается числом кэш-попаданий по отношению к общему числу обращений к памяти. Учитывая разницу в алгоритмах в режимах сквозной и обратной записи, эффективность использования кэш-памяти вычисляется выражениям (8.2) и (8.3) соответственно для сквозной и обратной записи.

Очевидно, эффективность работы системы с кэш-памятью будет зависеть не только от параметров кэш-памяти и выбранного алгоритма замещения, но и от класса решаемой задачи. Так, линейные программы должны хорошо работать с алгоритмами замещения типа очередь, а программы с большим числом условных переходов, зависящих от случайных входных данных, могут давать неплохие результаты с алгоритмами случайного замещения. Можно предположить, что программы, имеющие большое число повторяющихся участков (часто вызываемых подпрограмм и/или циклов) при прочих равных условиях обеспечат более высокую эффективность применения кэш-памяти чем линейные программы. И, разумеется, на эффективность напрямую должен влиять размер кэш-памяти.

Для проверки высказанных выше предположений выполняется настоящая лабораторная работа.

Задание

В данной лабораторной работе все варианты задания одинаковы: исследовать эффективность работы кэш-памяти при выполнении двух разнотипных программ, написанных и отлаженных вами при выполнении лабораторных работ № 2 и 4.

Порядок выполнения работы

Загрузить в модель учебной ЭВМ отлаженную программу из лабораторной работы № 2.

В меню Работа установить режим Кэш-память.

В меню Вид выбрать команду Кэш-память, открыв тем самым окно Кэш-память, в нем нажать первую слева кнопку на панели инструментов, открыв диалоговое окно Параметры кэш-памяти, и установить следующие параметры кэш-памяти: размер — 4, режим записи — сквозная, алгоритм замещения — случайное, без учета бита записи (W).

Запустить программу в автоматическом режиме; по окончании работы просмотреть результаты работы кэш-памяти в окне Кэш-память, вычислить значение коэффициента эффективности К и записать в ячейку табл. 9.15, помеченную звездочкой.

Выключить кэш-память модели (Работа | Кэш-память) и изменить один из ее параметров — установить флаг с учетом бита записи (в окне Параметры кэш-памяти).

Повторить п. 4, поместив значение полученного коэффициента эффективности в следующую справа ячейку табл. 9.15.

Последовательно меняя параметры кэш-памяти, повторить пп. 3—5, заполняя все ячейки табл. 9.15.

Совет

При очередном запуске программы не забывайте устанавливать процессор модели в начальное состояние, нажимая кнопку R в окне Процессор!

Повторить все действия, описанные в пп. 1—7 для программы из лабораторной работы, заполняя вторую таблицу по форме табл. 1

Содержание отчета

Две таблицы по форме табл. 9.15 с результатами моделирования программ из лабораторных работ № 2 и 4 при разных режимах работы кэш-памяти.

Выводы, объясняющие полученные результаты.

9.8.4. Контрольные вопросы

1 Как работает алгоритм замещения очередь при установленном флажке С учетом бита записи в диалоговом окне Параметры кэш-памяти?

2 Какой алгоритм замещения будет наиболее эффективным в случае применения кэш-памяти большого объема (в кэш-память целиком помещается программа)?

3 Как скажется на эффективности алгоритмов замещения учет значения бита записи W при работе кэш-памяти в режиме обратной записи? Сквозной записи?

4 Для каких целей в структуру ячейки кэш-памяти включен бит использования. Как устанавливается и сбрасывается этот бит?

Таблица 3. Результаты эксперимента

Способ	Сквозная запись					
Алгорит	Случайное		Очередь		Бит U	
Размер	безW	cW	без	с W	без	cW
4	*					
8						
16						
32						
4						
8						
16						

ЛАБОРАТОРНАЯ РАБОТА №14

ОРГАНИЗАЦИЯ ПАМЯТИ В КОМПЬЮТЕРАХ ТИПА IBM PC

Общие сведения об оперативной памяти

Оперативная память компьютера, иначе называемая оперативным запоминающим устройством (ОЗУ), используется для оперативного обмена информацией (командами и данными) между процессором, внешней памятью (например, дисковой) и различными подсистемами (видеоподсистема, подсистема ввода/вывода, коммуникации и т.д.). ОЗУ представляет собой память с произвольным доступом, поэтому для ее обозначения часто применяется аналогичное английское понятие RAM (Random Access Memory). Произвольность доступа подразумевает возможность операций чтения/записи с любой ячейкой ОЗУ в произвольном порядке. Основными требованиями, предъявляемыми к оперативной памяти являются следующие:

Большой (по меркам электронной памяти) объем, измеряемый десятками и сотнями мегабайт.

Быстродействие и производительность, позволяющие наиболее эффективно задействовать мощь современных процессоров.

Высокая надежность хранения данных.

Физически ОЗУ представляет собой просто набор последовательно расположенных ячеек, каждая из которых определяется собственным уникальным адресом. В сумме все ячейки оперативной памяти образуют адресное пространство (address space). В простейшем случае адресом некоторой ячейки памяти является просто ее номер относительно начала адресного пространства - нулевого адреса. В реальности же дело обстоит совсем иначе, поскольку по многим причинам программы не всегда могут использовать простейшую схему адресации оперативной памяти. Поэтому с точки зрения программиста наиболее важна не физическая, а логическая организация памяти. Логическая организация адресного пространства - это совокупность способов использования программами и операционными системами оперативной памяти компьютера.

Эффективное использование имеющегося ОЗУ является одной из главных задач, стоящих как перед программистами, так и перед пользователями, стремящимися получить максимальный выигрыш от использования персонального компьютера в своей деятельности. Поэтому для эффективного использования памяти необходимо четкое понимание принципов ее логической организации.

Распределение памяти

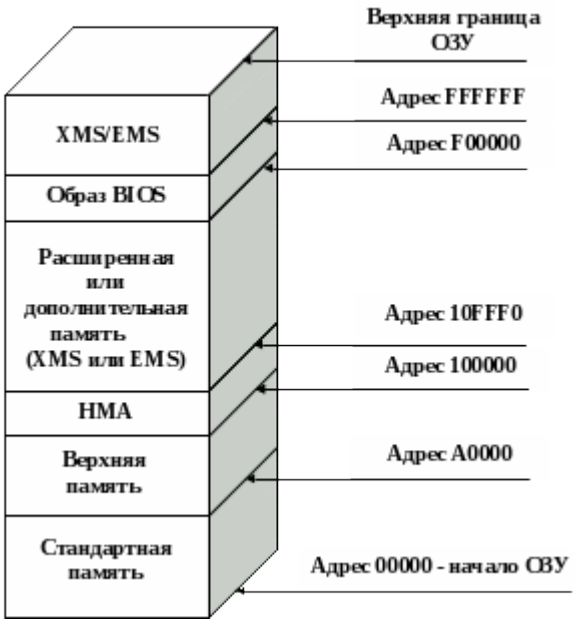
С точки зрения использования программами адресного пространства его можно подразделить на следующие участки:

стандартная память (Conventional Memory)

расширенная (Extended Memory), либо дополнительная (Expanded Memory), память

верхняя память (High Memory или Upper Memory)

помимо перечисленных категорий следует также особо рассмотреть так называемую область верхней памяти (High Memory Area – HMA), расположенную сразу за пределами 1 Мб, а также области теневой памяти (Shadow Memory). Распределение памяти представлено на рис. 1.



16 Мб

15 Мб

1088 Кб

1024 Кб

640 Кб

0 Кб

Рис 1. Распределение памяти в IBM PC

Стандартная память

По причинам исторического характера программы, являющиеся совместимыми с операционной системой MS-DOS и ее клонами, могут использовать в своей работе только первый мегабайт из всей памяти, установленной на компьютере, если не применяются

специальные средства поддержки. Для объяснения этого обстоятельства следует обратить внимание на особенности архитектуры процессора Intel 8086 – базового процессора семейства 80x86. В процессоре 8086 16 ножек (pins) посылают сигналы, соответствующие 16 битам текущих данных, которыми процессор обменивается с системным ОЗУ. Однако данные не имеют никакого смысла сами по себе, если нет возможности следить за тем, что они собой представляют. Системная шина должна знать, куда направляются конкретные данные или откуда они поступили. Для выполнения этой задачи процессор использует еще 20 из своих ножек для создания уникальных адресов памяти. Это дает возможность процессору семейства 8086/8088 адресовать 2 в степени 20 уникальных байтов, что и соответствует 1 мегабайту памяти. В оригинальной архитектуре персонального компьютера 640 Кб из этого 1 Мб было зарезервировано под DOS и прикладные программы, работающие под ее управлением, а область с 640 Кб до 1 Мб была зарезервирована для системного пользования. Большая часть 640 килобайт обычной памяти используется почти постоянно, но в области служебных адресов существуют участки, которые система не использует и которые могут быть доступны для других целей. Участки системной памяти, расположенные между 640 килобайт и 1 мегабайт интенсивно используются программами-расширителями памяти, такими, как Microsoft EMM386, Quarterdeck QEMM и Qualitas 386MAX.

Предварительно следует дать необходимые понятия о разделении адресного пространства в пределах 1 Мб на регионы. Общепринято делить 1 Мб памяти на 16 последовательных участков по 64 Кб каждый. Эти участки, помечаются шестнадцатеричными целыми числами от 0 до F, т.е., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Из этих регионов DOS получает в свое распоряжение области с номерами 0 - 9 (10 первых участков по 64 Кб, итого 640 Кб), а остальные 6 участков памяти, расположенные между 640 Кб и 1 Мб (участки с номерами A, B, C, D, E, F), относятся к области служебных адресов и поступают в распоряжение системы. Удобство подобного разделения заключается в том, что номеру какого-либо 64Кб-региона может быть легко поставлен в соответствие адрес этого региона в адресном пространстве 1 Мб путем добавления 3 цифр справа к номеру региона.

Рассмотрим это на конкретном примере. Итак, регион A находится в адресном пространстве непосредственно за областью в 640 Кб, то есть, говоря иными словами, адрес его должен превышать 655359 байт (учитывая, что отсчет адресов идет с 0, число 655359 как раз соответствует 640 Кб) в десятичной нотации или 9FFFF в шестнадцатеричной. С учетом вышесказанного, адрес региона с номером A в адресном пространстве будет выглядеть как A000. С точки зрения процессора, этот адрес будет выглядеть как A0000, т.е A000 × 16 (10 в шестнадцатеричной нотации). Это объясняется тем, что адрес, задаваемый в программе, может быть выражен 16-разрядным числом, т.е. содержать не более 4-х шестнадцатеричных цифр.

Адресная шина процессора имеет 20 разрядов, поэтому перед вычислением адреса он переводится из 16-разрядной формы в 20-разрядную путем добавления справа еще одной шестнадцатеричной цифры, что эквивалентно умножению на 10 в шестнадцатеричной нотации или на 16 – в десятичной. Выполнив теперь перевод полученного адреса региона A – A0000 в десятичную форму получим число 655360, что доказывает, что регион с номером A находится как раз над границей памяти DOS в 640 Кб.

Выполнив аналогичные вычисления для региона с номером C, получим следующие результаты:

Добавим 3 шестнадцатеричных цифры справа к номеру региона, получим C000.

Для вычисления адреса по правилам процессора 8086 умножим этот адрес на 16: C000 × 16 = C0000.

Переведем его в десятичную форму: C0000 = 786432

Посчитаем сумму адресов, занимаемых предыдущими 12 регионами (0 - B): 12 * 65536 = 786431 (отсчет идет с 0!).

Таким образом, вновь доказана корректность вычислений и удобство принятого обозначения регионов. Каждый 64-килобайтный регион может быть дополнительно поделен на 16 областей размером в 4 Кб, которые также помечаются шестнадцатеричными числами. Например, область A разбивается на дополнительные области от 0 до F: A0, A1, ... , AF. В отношении вычисления адресов этих дополнительных областей действует то же правило, что и для регионов, с той разницей, что к номеру области добавляются 2 цифры, а не 3, как в случае региона. Например, адрес области A7 может быть вычислен следующим образом:

Добавляем 2 цифры к номеру области: A700.

Переводим адрес в эффективный адрес процессора: A700 × 16 = A7000.

Вычисляем его десятичное значение: A7000 = 684032.

Легко проверить, что если к 640 Кб, которые занимают 10 регионов, идущие перед регионом A, добавить адреса, занимаемые 7 4-килобайтными областями внутри региона A (A0 – A6), то получится:

655359 + 4096 * 7 = 684031

Все приведенные выше сведения, возможно, грешат излишней детальностью, но являются, тем не менее, исключительно важными для понимания дальнейшего изложения.

Расширенная память

Появление расширенной памяти (Extended Memory Specification – XMS) было обусловлено появлением процессора 80286, который имеет на четыре адресных линии больше, чем 8086/8088, что позволяет адресоваться к количеству адресов, большему в 16 раз (2 в степени 4), т.е. к 16 Мб. Адреса памяти выше предела 1MB и называются расширенной

памятью. Характерной особенностью процессора Intel 80286 является возможность работы в 2-х режимах: в реальном режиме, в котором 80286 работает как более производительный 8086, и в защищенном, позволяющем адресовать память, лежащую за пределами 1-мегабайтного барьера. Работая в реальном режиме, 80286 не может получить доступа к расширенной памяти и по-прежнему ограничен 1 Мб адресуемого пространства. Чтобы использовать расширенную память, он должен работать в режиме виртуальной (защищенной) адресации. Изначально разработчики этого процессора не предусмотрели простого способа переключаться обратно в режим реальной адресации из режима виртуальной адресации (защиты). Это затруднило использование расширенной памяти в программах и она использовалась в основном под буферы ввода – вывода и печати. Однако, впоследствии, благодаря появлению специальных программ-расширителей – драйверов расширенной памяти – эта проблема была решена. Драйверы расширенной памяти обеспечивают полную поддержку расширенной памяти и позволяют программам осуществлять доступ к ней, а также выполнять программы из расширенной памяти. Наиболее популярными драйверами расширенной памяти являются HIMEM.SYS фирмы Microsoft и DOSHI.SYS фирмы Quarterdeck, входящий в состав пакета QEMM.

На компьютере с микропроцессором 80286, имеющем 24 адресных линии, можно получить до 15MB расширенной памяти (весь объем памяти составляет 16MB). Микропроцессоры 80386 и выше физически способны адресовать 4 гигабайта памяти, имея 32 адресные линии (2 в степени 32 байтов).

Дополнительная память

Дополнительная память (Expanded memory), часто называемая EMS (Expanded Memory Specification) или LIM (согласно ее разработчикам: Lotus, Intel и Microsoft), представляет собой способ доступа к памяти, лежащей за пределами 1 Мб, посредством окна размером 64 Кбайт, лежащего в области служебных адресов между 640 Кб и 1 Мб. При этом используется так называемая схема коммутации банков памяти (bankswitching) – способ управления памятью, когда физическая память разбита на несколько сегментов (банков) длиной, равной размеру адресного пространства процессора. В каждый момент процессор работает с одним банком. Память EMS логически подразделяется на участки размером по 16 Кбайт, называемые страницами. В процессе работы с дополнительной памятью часть содержимого банка размером в 4 страницы, или в 64 Кб отображается в определенный регион, лежащий в пределах 1 Мб, чаще всего в регион E. Данный регион принято называть EMS-фреймом. EMS-фрейм как бы «скользит» по всему адресному пространству банка, что позволяет программам, физически обращаясь к адресам в пределах 1 Мб, получать, тем не менее, доступ ко всей памяти, которая установлена на компьютере.

Дополнительная память особенно полезна потому, что, в отличие от расширенной памяти, она использует только адреса ниже границы в 1MB, следовательно, она может использоваться на машинах с процессорами 8086/8088, кроме того, при программировании нет необходимости использовать адреса, лежащие за пределами 1 Мб. Их отображение в EMS-фрейм будет выполнено драйвером дополнительной памяти. Недостатком EMS-памяти является некоторое замедление при обращении к памяти, которое объясняется накладными расходами, связанными с отображением страниц EMS в EMS-фрейм, переключением банков и т.д. Основными драйверами, обеспечивающими доступ к дополнительной памяти, являются EMM386 (Microsoft), QEMM (Quarterdeck) и 386MAX (Qualitas).

Область верхней памяти (HMA)

Начиная с процессора 80286 можно было наблюдать интересный эффект, связанный с тем, как происходит адресация в пределах 1 Мб. Согласно используемой при программировании в реальном режиме сегментной модели, все адресное пространство может быть представлено в виде пересекающихся между собой участков – сегментов размером в 64 Кб, которые должны начинаться с адреса, кратного 16. До тех пор, пока сегмент начинается с адресов, лежащих до 960 Кб включительно, все более или менее понятно. Однако, как быть в случае, когда сегмент начинается с адреса, лежащего выше 960 Кб? Ведь в этом случае все 64 Кб, входящие в сегмент, просто не уместятся в пределах 1 Мб. В случае с процессором 8086 просто-напросто происходил циклический возврат к началу. Например, если сегмент начинался, скажем, по адресу FB00 или 1028096 байт (1 Мб = 1048576 байт), то первые 4FFF или 20480 байт сегмента располагались в самом верху адресного пространства до 1 Мб включительно, а остальные B000 или 45056 байт – начиная опять с 0 адреса, то есть с самого начала адресного пространства процессора. В

процессоре 80286 ситуация изменилась, поскольку стало возможным адресовать физически более 1 Мб ОЗУ. Таким образом, когда процессор 80286 работает как 8086 и пытается получить доступ к одному из сегментов, лежащих близко к вершине адресного пространства, то адрес, полученный в результате операции будет лежать за пределами 1 Мб и, следовательно, возникает возможность обращаться к 64-килобайтной области (точнее, к 64 Кб минус 16 байт), лежащей за пределами 1 Мб, причем делать это в режиме реальной адресации, без каких-либо средств расширения DOS. Адресуемая подобным образом область размером почти в 64 Кб, лежащая в диапазоне 1024 Кб – 1088 Кб, получила название области верхней памяти (High Memory Area – HMA). В настоящее время типичным применением области HMA стало размещение в ней ядра DOS, что позволяет освободить дополнительно часть памяти ниже 640 Кб для пользовательских программ.

Чтобы ликвидировать разногласия в случаях, когда микропроцессор 80286 работает как микропроцессор 8086, в шину были встроены специальные средства (так называемый мост A20 – A20 Gate), которые заставляют память в режиме реального времени переходить циклически на низшие адреса, как это происходит в 8086/8088. Одним из применений драйвера HIMEM.SYS является отмена действия моста A20, что позволяет программам, написанным особым образом, обращаться к области HMA.

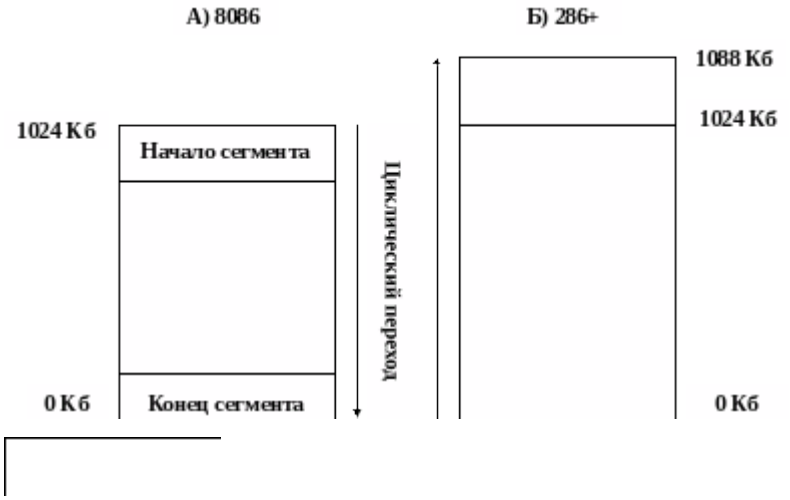


Рис 2. Особенности адресации в процессорах Intel 8086 (а) и Intel 80286 и старших (б)

Верхняя память

Память в высших адресах или верхняя память (High Memory) – это область объемом 384 Кбайт, расположенная между границами 640 Кбайт и 1 Мбайт (адреса A000 до FFFF), зарезервированная IBM для системного аппаратного обеспечения – для видеопамяти, BIOS и прочего. Во многих случаях она, однако, используется не полностью, и в ней образуются «дыры» – свободные участки, которые не используются в служебных целях, но к которым не имеют прямого доступа программы, работающие в 640 Кб основной памяти. Посредством расширителей памяти – таких как EMM386 (производства Microsoft), QEMM (производства Quarterdeck) и 386MAX (производства Qualitas) можно управлять этой областью зарезервированной памяти и перенести туда резидентные программы из основной памяти, тем самым освобождая больше обычной памяти под прикладные программы.

Структура верхней памяти

Для эффективного использования памяти, расположенной в верхних адресах, необходимо, прежде всего, знать, какие именно регионы этой памяти, используются всегда, а какие – зачастую остаются свободными.

Как уже упоминалось, верхняя память представлена 384 Кб, расположенными в адресном пространстве между 640 Кб и 1 Мб и включает в себя рассмотренные ранее регионы A, B, C, D, E и F. Следует сразу же заметить, что не существует четких стандартов на использование всех этих регионов. В общем случае использование участков верхней памяти

определяется конкретной конфигурацией и может быть разным на разных машинах. В то же время существуют определенные рассматриваемые ниже соглашения об их распределении.

Регионы A и B зарезервированы для видеопамяти, но в большинстве случаев по крайней мере, часть их не используется. Оригинальный монохромный адаптер берет 4 Кбайт из региона B, чтобы разместить 4000 байт, необходимых для описания текстового экрана (25 строк на 80 столбцов, по 2 атрибута). Эта память начинается с адреса B000 и продолжается почти до адреса B100, хотя возможно использование области вплоть до B200 и даже B400 в зависимости от конкретной BIOS. В случае наличия в системе адаптера CGA для организации видеопамяти используется 16 Кбайт, лежащих в адресах от B800 до BC00. Хотя это оставляет остаток региона B в отрезке BC00h - C000h при наличии CGA в принципе свободным, им можно пользоваться только на свой риск, поскольку некоторые программы предполагают, что эта неиспользуемая видеопамять доступна и прекрасным образом испортят все, что программа управления памятью поместит в эту область. Адаптеры EGA и VGA используют область B800 - C000 для организации видеопамяти текстового режима и тех графических режимов, которые подражают CGA, а также всю область A для графики с высокой разрешающей способностью.

Другой стандартной областью является область F. В ней располагаются процедуры и данные BIOS и некоторая другая важная системная информация. Тем не менее, не все системы полностью занимают весь отведенный участок размером 64 Кб, и не все из этих 64 Кб нужны после загрузки. Например, в некоторых системах память от F000 до F800 используется программой установки и диагностики, которая может быть вызвана во время загрузки. Поскольку этот участок не используется после загрузки, можно позволить программе управления памятью разместить там что-нибудь другое.

Области C и D запутаны больше всего. В системах, имеющих EGA- и VGA-адаптеры, видео-ПЗУ обычно находится на дне области C. Остаток региона C, не используемый для размещения видеоПЗУ (адреса выше C800), а также вся область D применяются для размещения драйверов и прочего программного обеспечения, обеспечивающего поддержку всевозможного специфического оборудования – сетевых карт, адаптеров SCSI и т.д.

Наконец, регион E в случае наличия в системе диспетчера дополнительной (EMS) памяти используется как EMS-фрейм.

Подытоживая все эти сведения, можно заметить, что в среднестатистической системе гарантированно занятыми являются только регионы A (видеопамять графического режима для адаптеров EGA, VGA и выше) и F (системная BIOS), кроме того, почти всегда (при наличии цветного адаптера) занята область размером 16 Кб, лежащая в адресах B800 – BC00 (видеопамять текстового режима), и область размером в 32 Кб в начале региона C в адресах C000 – C800 (видеоПЗУ). Занятость же прочих регионов верхней памяти в общем случае не

определена. Таким образом, стандартно занято 64 Кб + 64 Кб + 16 Кб + 32 Кб = 176 Кб верхней памяти. Прочие 208 Кб, вообще говоря, могут быть свободны и использованы для хранения программ, которые обычно располагаются в основной памяти. Реально, во многих случаях занятым оказывается еще и регион E, содержащий EMS-фрейм, а вторая половина региона C и регион D, напротив, чаще всего свободны. В силу данного обстоятельства в верхней памяти обычно остаются неиспользованными как минимум 96 Кб.

Разумеется, хранение в верхней памяти прикладных программ, работающих в адресном пространстве 640 Кб, было бы сопряжено со слишком большими трудностями, однако, если речь идет о драйверах, которые не должны постоянно выгружаться и загружаться в память, а должны находиться в ней резидентно, то использование верхней памяти в данном случае имеет большие преимущества. Поэтому была изобретена технология, позволяющая переносить драйверы, расположенные в обычной памяти, в свободные блоки верхней памяти, освобождая таким образом основное ОЗУ. Блоки верхней памяти, используемые подобным образом, получили название Upper Memory Blocks – UMB. Организацией блоков UMB и переносом в них драйверов из нижней памяти занимаются все те же менеджеры памяти, о которых уже шла речь выше – QEMM, EMM386, 386MAX и т.д.

В качестве примера, иллюстрирующего преимущество правильного использования верхней памяти и знания структуры памяти вообще, можно привести следующий факт. Если после загрузки DOS и необходимых драйверов устройств остаются свободными примерно 550-570 Кб ОЗУ, то при помещении ядра DOS в HMA с помощью драйвера HIMEM.SYS (или DOSHI.SYS), а также драйверов устройств в UMB посредством EMM386 или QEMM доступными для прикладных программ оказываются уже 620-635 Кб (а иногда – и больше). Таким образом, правильное конфигурирование системы и настройка менеджеров памяти позволяет буквально из ничего выжать дополнительные 50-85 Кб (удавалось получать даже 100 Кб)!

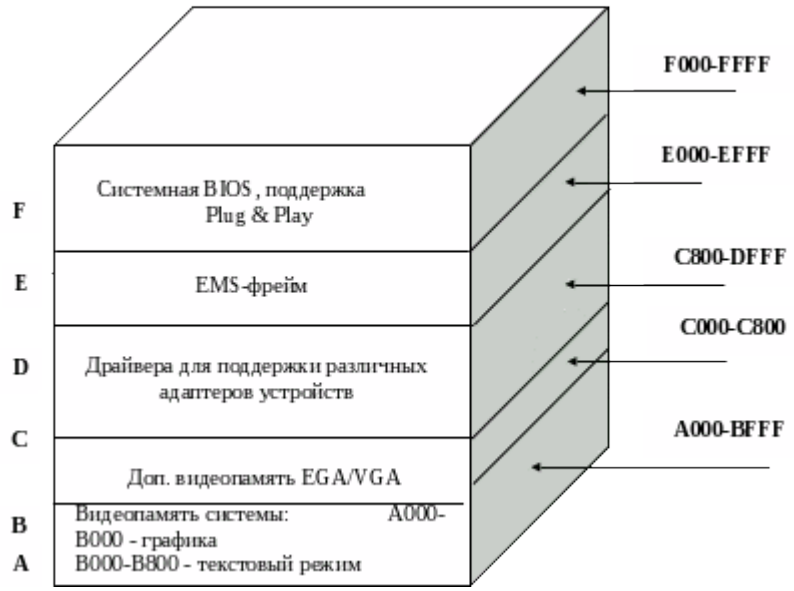


Рис 3. Структура блоков верхней памяти (UMB)

Теневая память

Теневая память (Shadow Memory) ускоряет обращение к обычно медленной постоянной памяти, такой как System BIOS или VideoROM BIOS за счет ее подмены быстрой оперативной памятью, имеющей более высокую разрядность. Теневая память подразделяется на теневую RAM (Shadow RAM) и теневую ROM (Shadow ROM). При инициализации Shadow ROM содержимое затеняемой области копируется в область теневой памяти и при дальнейшем обращении к затеняемым участкам памяти вместо реального ROM подставляется участок Shadow RAM, запись же в эту область и вовсе блокируется. Shadow ROM отличается тем, что при записи в ROM запись происходит как собственно в ROM, так и в участок затемняющей его теневой памяти. Области, используемые для затемнения постоянной памяти находятся в верхней памяти. Как правило Shadow Memory располагается в регионах C и D и представляет собой, чаще всего, копию содержимого ПЗУ различных адаптеров для убыстрения доступа к ним. Обычно включение или выключение теневой памяти осуществляется через BIOS Setup компьютера. Эту же функцию имеют и некоторые менеджеры памяти, в частности EMM386.

Особенности адресации в защищенном режиме и в среде Windows

Работа в защищенном режиме процессоров Intel, начиная с 386, имеет значительные отличия в силу используемых методов адресации памяти. Современные процессоры поддерживают концепцию т.н. виртуальных адресных пространств. Иначе говоря, все адресное пространство с точки зрения программы представляется непрерывным участком размером до 4 Гб. Эта особенность, основанная на 32-разрядной адресации памяти в этих процессорах, позволяет избежать всевозможных неудобств, связанных с сегментацией, и значительно упрощает и убыстряет обращение к памяти, поскольку требуется меньше времени для вычисления адреса (по сути дела вычисляется только смещение). Вместе с тем, указанная

схема адресации имеет свои сложности. Во-первых, сам по себе виртуальный адрес в пределах 4 Гб ничего не значит, если не известно адресное пространство, которому он принадлежит (а их может быть до 16384!). Во-вторых, поскольку объем ОЗУ на современных компьютерах, как правило, все еще сильно отстает от 4 Гб, то, даже вычислив адрес и определив его адресное пространство (контекст), необходимо определить, находится ли он в физическом ОЗУ или выгружен на диск в файл подкачки. Как правило, развитые современные операционные системы типа Windows содержат в составе своего ядра специальные компоненты, такие как, например, диспетчер защищенного режима, которые неявно обрабатывают все ситуации, связанные с вычислением адресов в защищенном режиме. Таким образом, для программ, предназначенных для работы в Windows, не существует понятия сегментации, а также описанного в предыдущих разделах распределения адресного пространства в пределах 1 Мб.

При работе в режиме MS-DOS программы также могут в некоторых случаях получать доступ к 4 Гб памяти, используя схему адресации защищенного режима. Это выполняется либо с помощью драйверов поддержки расширенной памяти, либо с помощью т.н. программ-расширителей DOS (DOS Extenders). Во втором случае расширитель DOS образует как бы 32-разрядную надстройку над 16-разрядной DOS и обеспечивает тем самым возможность 32-битной адресации и поддержку защищенного режима работы, которую невозможно реализовать обычными средствами DOS. Прикладная программа работает в этом случае под управлением расширителя DOS, ничего не знает о тонкостях адресации и считает, что ей доступно до 4 Гб памяти – все остальное делает расширитель. Альтернативным и гораздо более распространенным способом адресации больших блоков памяти в мире MS-DOS служит использование расширенной памяти. Как указывалось ранее, драйверы расширенной памяти, подобные HIMEM.SYS или DOSHI.SYS, обеспечивают программам доступ к памяти за пределами 1 Мб. Механизм их работы заключается в том, что при обращении к адресу, лежащему за пределами 1 Мб, драйвер на мгновение переводит компьютер в защищенный режим, выполняет обращение к участку памяти, а затем возвращает компьютер обратно в реальный режим, обеспечивая тем самым возможность программам обращаться к памяти, лежащей выше 1 Мб.

Вопросы по теории

Понятие о регионах. Вычисление адресов регионов.

Охарактеризовать применение расширенной и дополнительной памяти.

Структура верхней памяти.

Область HMA и особенности адресации процессоров 8086 и 80286.

Особенности адресации в защищенном режиме.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Для получения информации о текущем распределении памяти используются программы для просмотра состояния памяти. Некоторые из этих программ достаточно просты и позволяют получить информацию лишь самого общего характера. Для более полного обзора необходимо применять специализированные утилиты, которые часто входят в пакеты различных диагностических программ и менеджеров памяти (например, программу Quarterdeck Manifest из пакета Quarterdeck QEMM).

Просмотр состояния памяти посредством утилиты MEM

Утилита MEM, несмотря на свои достаточно ограниченные возможности, позволяет получить подробные сведения о распределении памяти. Ее важнейшим достоинством является доступность, так как она входит в комплект поставки Windows '95/98 и практически всегда находится в каталоге COMMAND системного каталога Windows. Так, например, если Windows установлена в каталог C:\WINDOWS, то MEM находится в каталоге C:\WINDOWS\COMMAND. Таким образом, использование MEM - это простейший способ получить информацию о состоянии памяти системы.

В простейшем случае утилита MEM запускается без параметров и в этом режиме выдает всю основную информацию, сведенную в несколько таблиц. Самая первая таблица показывает информацию о программных модулях, расположенных в стандартной памяти с указанием их размера. Вторая таблица представляет собой карту использования адресного пространства, в которой указаны объемы используемой стандартной и расширенной памяти. В последних строчках вывода MEM показан ряд дополнительных сведений, в частности, указано местонахождение ядра DOS (в нижней памяти или в HMA). Для получения некоторых дополнительных сведений можно использовать различные ключи при запуске MEM. Чаще всего используются ключи /C, /D, /F и /P. Ключ /P включает режим постраничного вывода и может быть использован в сочетании с любыми другими ключами. Прочие ключи являются взаимоисключающими. Использование ключа /C приводит к выводу той же информации, что и в режиме без параметров. Ключ /F показывает свободные участки памяти, а ключ /D используется для получения наиболее полной информации о распределении памяти, которая может быть использована в целях отладки. Примеры запуска:

MEM /P - постраничный вывод информации о распределении памяти

MEM /D /P - вывод детальных сведений о распределении памяти

Задание к лабораторной работе

Цель работы: закрепить знания о логической организации памяти, получить навыки использования специализированных программ для получения сведений о распределении памяти, исследовать влияние менеджеров памяти на ее распределение.

Работая в Windows '9x, выполнить запуск утилиты MEM, рассмотреть и проанализировать выводимые сведения.

Перезагрузиться в режиме MS-DOS, выполнить запуск MEM, проанализировать текущее распределении памяти.

Выполнить полную перезагрузку компьютера, по нажатию клавиши F5 в момент загрузки отключить драйвера. Исследовать распределение памяти

Практическая работа №14-15.

Тема: Программирование и отладка программ. Создание и разработка программы на ASSEMBLER. Этапы компиляции исходного кода в машинные коды и способы отладки. Использование отладчиков.

Цель работы : Практическое освоение основных функций TURBO DEBUGGER.

1.Краткие теоретические сведения.

Турбо отладчик (Turbo Debugger) - это современный отладчик, позволяющий отлаживать программы на уровне исходного текста и предназначенный для программистов, работающих на Турбо языках фирмы Borland. Много-численные перекрывающие друг друга окна, а также сочетание спускающихся и раскрывающихся меню обеспечивают быстрый, интерактивный пользовательский интерфейс. Интерактивная, контекстно-зависимая система подсказки обеспечит вас подсказкой на всех стадиях работы. Непосредственно после запуска отладчика открыто окно CPU. В окне CPU (ЦП) показано все состояние центрального процессора. С его помощью вы можете проверять и изменять биты и байты, составляющие код и данные программы. В окне Code (Код) для временной коррекции своей программы вы можете использовать встроенный Ассемблер. При этом инструкции вводятся точно также, как при наборе исходных операторов Ассемблера. Можно также получить доступ к соответствующим данным любой структуры данных, выводя и изменяя их в различных форматах.

В области регистров (верхняя область справа от области кода) выводится содержимое регистров центрального процессора.

Верхней правой областью является область флагов, где показано содержимое восьми флагов центрального процессора. В области флагов показано значение каждого флага ЦП.

В области данных показано непосредственное содержимое выбранной области памяти. В левой части каждой строки показан адрес данных, выводимых на данной строке. Адрес выводится в виде шестнадцатеричного значения сегмента и смещения. Значение сегмента заменяется именем сегмента DS, если значение сегмента совпадает с текущим содержимым регистра DS.

В правой части каждой строки выводятся символы, соответствующие по казанным байтам. Турбо отладчик выводит все печатаемые значения, соответствующие байтовым эквивалентам, поэтому не удивляйтесь, если на экране вы увидите странные символы - просто это символьный эквивалент шестнадцатеричных значений байтов данных.

В нижнем правом углу окна CPU показано содержимое стека

2.Задания для самостоятельной работы.

Задание 1 Используя TURBO DEBUGGER, ассемблировать исходный текст программы HELLO.ASM в режиме /zi , произвести его компоновку в режиме /v . Поместить полученные файлы в том же каталоге, что и TD.EXE.

Задание 2.Запустить программу TD на выполнение. После появления визитной карточки отладчика нажать клавишу ENTER. Обратить внимание на то, что в нижней строке расположена подсказка о назначении функциональных клавиш, в верхней строке перечислены меню отладчика, а в основном поле экрана открыто окно CPU (ЦП). Клавишей ZOOM изменить размер открытого окна.

Используя клавишу F10, перейти в главное меню. Открыть окно FILE. Включить режим OPEN... . Клавишей TAB выделить окно FILES. Курсорными клавишами выбрать имя файла hello.exe и загрузить его. Сравните информацию, содержащуюся в рамке кода с листингом вашей программы.

Задание 3.В окне CPU произвести трассировку программы (пошаговое выполнение) нажатием клавиши F8. На каждом шаге контролировать содержимое регистров, флагов и состояние стека. После завершения программы перейти в окно WINDOW главного меню и установить режим USER SCREEN. Убедитесь, что программа выполнила свою задачу. Клавишей ESC верните изображение окна CPU

Практическая работа №18-19

Тема: FASM (Flat Assembler).

1 ЦЕЛЬ РАБОТЫ

Приобретение навыков работы в FASM (Ассемблер)

2 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Прочитать задания к работе

Оформите отчет, который должен содержать:

–титульный лист (см. приложение);

–постановку задачи;

- описание пошагового исполнения;

– отчет о полученном результате

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ

FASM (Flat Assembler). Этот компилятор достаточно прост в установке и использовании, отличается компактностью и быстротой работы, имеет богатый и емкий макросинтаксис, позволяющий автоматизировать множество рутинных задач. Его последнюю версию вы можете скачать по адресу: сайт выбрав flat assembler for Windows. Чтобы установить FASM, создайте папку, например, "D:\FASM" и в нее распакуйте содержимое скачанного zip-архива. Запустите FASMW.EXE и закройте, ничего не изменяя. Кстати, если вы пользуетесь стандартным проводником, и у вас не отображается расширение файла (например, .EXE), рекомендую выполнить Сервис -> Свойства папки -> Вид и снять птичку с пункта Скрывать расширения для зарегистрированных типов файлов. После первого запуска компилятора в нашей папке должен появиться файл конфигурации — FASMW.INI. Откройте его при помощи стандартного блокнота и допишите в самом низу 3 строчки:

[Environment]

Fasminc=D:\FASM\INCLUDE

Include=D:\FASM\INCLUDE

Если вы распаковали FASM в другое место — замените "D:\FASM\" на свой путь. Сохраните и закройте FASMW.INI. Забегая вперед, вкратце объясню, как мы будем пользоваться компилятором:

1. Пишем текст программы, или открываем ранее написанный текст, сохраненный в файле .asm, или вставляем текст программы из буфера обмена комбинацией.

2. Жмем F9, чтобы скомпилировать и запустить программу, или Ctrl+F9, чтобы только скомпилировать. Если текст программы еще не сохранен — компилятор попросит сохранить его перед компиляцией.

3. Если программа запустилась, тестируем ее на правильность работы, если нет — ищем ошибки, на самые грубые из которых компилятор нам укажет или тонко намекнет. Ну, а теперь мы можем приступить к долгожданной практике. Запускаем наш FASMW.EXE и набираем в нем код нашей первой программы:

```
        include '%fasminc%/win32ax.inc'

        .data
Caption db 'Моя первая программа.',0
        Text db 'Всем привет!',0
        .code
        start:
        invoke MessageBox,0,Text,Caption,MB_OK
        invoke ExitProcess,0

        .end start
```

Жмем Run -> Run, или F9 на клавиатуре. В окне сохранения указываем имя файла и папку для сохранения. Желательно привыкнуть сохранять каждую программу в отдельную папку, чтобы не путаться в будущем, когда при каждой программе может оказаться куча файлов: картинки, иконки, музыка и прочее. Если компилятор выдал ошибку, внимательно перепроверьте указанную им строку — может, запятую пропустили или пробел. Также необходимо знать, что компилятор чувствителен к регистру, поэтому .data и .Data воспринимаются как две разные инструкции. Если же вы все правильно сделали, то результатом будет простейший MessageBox (рис. 1). Теперь давайте разбираться, что же мы написали в тексте программы. В первой строке директивой include мы включили в нашу программу большой текст из нескольких файлов. Помните, при установке мы прописывали в фасмовский ини-файл 3 строчки? Теперь %fasminc% в тексте программы означает D:\FASM\INCLUDE или тот путь, который указали вы. Директива include как бы вставляет в указанное место текст из другого файла. Откройте файл WIN32AX.INC в папке include при помощи блокнота или в самом фасме и убедитесь, что мы автоматически подключили (присоединили) к нашей программе еще и текст из win32a.inc, macro/if.inc, кучу непонятных (пока что) макроинструкций и общий

набор библиотек функций Windows. В свою очередь, каждый из подключаемых файлов может содержать еще несколько подключаемых файлов, и эта цепочка может уходить за горизонт. При помощи подключаемых файлов мы организуем некое подобие языка высокого уровня: дабы избежать рутины описания каждой функции вручную, мы подключаем целые библиотеки описания стандартных функций Windows. Неужели все это необходимо такой маленькой программе? Нет, это — что-то вроде "джентльменского набора на все случаи жизни". Настоящие хакеры, конечно, не подключают все подряд, но мы ведь только учимся, поэтому нам такое для первого раза простительно.

Далее у нас обозначена секция данных — .data. В этой секции мы объявляем две переменные — Caption и Text. Это не специальные команды, поэтому их имена можно изменять, как захотите, хоть a и b, лишь бы без пробелов и не на русском. Ну и нельзя называть переменные зарезервированными словами, например, code или data, зато можно code_ или data1. Команда db означает "определить байт" (define byte). Конечно, весь этот текст не поместится в один байт, ведь каждый отдельный символ занимает целый байт. Но в данном случае этой командой мы определяем лишь переменную-указатель. Она будет содержать адрес, в котором хранится первый символ строки. В кавычках указывается текст строки, причем кавычки по желанию можно ставить и 'такие', и "такие" — лишь бы начальная кавычка была такая же, как и конечная. Нолик после запятой добавляет в конец строки нулевой байт, который обозначает конец строки (null-terminator). Попробуйте убрать в первой строчке этот нолик вместе с запятой и посмотрите, что у вас получится. Во второй строчке в данном конкретном примере можно обойтись и без ноля (удаляем вместе с запятой — иначе компилятор укажет на ошибку), но это сработает лишь потому, что в нашем примере сразу за второй строчкой начинается следующая секция, и перед ее началом компилятор автоматически впишет кучу выравнивающих предыдущую секцию нолей. В общих случаях ноли в конце текстовых строк обязательны! Следующая секция — секция исполняемого кода программы — .code. В начале секции стоит метка start:. Она означает, что именно с этого места начнет исполняться наша программа. Первая команда — это макроинструкция invoke. Она вызывает встроенную в Windows API-функцию MessageBox. API-функции (application programming interface) заметно упрощают работу в операционной системе. Мы как бы просим операционную систему выполнить какое-то стандартное действие, а она выполняет и по окончании возвращает нам результат проделанной работы. После имени функции через запятую следуют ее параметры. У функции MessageBox параметры такие:

1-й параметр должен содержать хэндл окна-владельца. Хэндл — это что-то вроде личного номера, который выдается операционной системой каждому объекту (процессу,

окну и др.). 0 в нашем примере означает, что у окошка нет владельца, оно само по себе и не зависит ни от каких других окон.

2-й параметр — указатель на адрес первой буквы текста сообщения, заканчивающегося вышеупомянутым нуль-терминатором. Чтобы наглядно понять, что это всего лишь адрес, сместим этот адрес на 2 байта прямо в вызове функции: invoke MessageBox,0,Text+2,Caption,MB_OK и убедимся, что теперь текст будет выводиться без первых двух букв.

3-й — указатель адреса первой буквы заголовка сообщения.

4-й — стиль сообщения. Со списком этих стилей вы можете ознакомиться, например, в INCLUDE\EQUATES\ USER32.INC. Для этого вам лучше будет воспользоваться поиском в Блокноте, чтобы быстро найти MB_OK и остальные. Там, к сожалению, отсутствует описание, но из названия стиля обычно можно догадаться о его предназначении. Кстати, все эти стили можно заменить числом, означающим тот, иной, стиль или их совокупность, например: MB_OK + MB_ICONEXCLAMATION. В USER32.INC указаны шестнадцатеричные значения. Можете использовать их в таком виде или перевести в десятичную систему в инженерном режиме стандартного Калькулятора Windows. Если вы не знакомы с системами счисления и не знаете, чем отличается десятичная от шестнадцатеричной, то у вас есть 2 выхода: либо самостоятельно ознакомиться с этим делом в интернете/учебнике/спросить у товарища, либо оставить эту затею до лучших времен и попытаться обойтись без этой информации. Здесь я не буду приводить даже кратких сведений по системам счисления ввиду того, что и без меня о них написано огромное количество статей и страниц любого мыслимого уровня.