Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Информатика»

Н. В. Самовендюк

АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

ПРАКТИКУМ
по выполнению лабораторных работ для студентов специальности 1-40 04 01 «Информатика и технологии программирования» дневной формы обучения

Гомель 2020

УДК 004.42(075.8)
ББК 32.973.22я73
С17

Рекомендовано научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(*протокол № 7 от 04.03.2019 г.*)

Рецензент: доц. каф. «Промышленная электроника» ГГТУ им. П. О. Сухого
канд. техн. наук *А. В. Ковалев*

Самовендюк, Н. В.

С17 Архитектура вычислительных систем : практикум по выполнению лаборатор. работ для студентов специальности 1-40 04 01 «Информатика и технологии программирования» днев. формы обучения / Н. В. Самовендюк. – Гомель : ГГТУ им. П. О. Сухого, 2020. – 95 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: https://elib.gstu.by. – Загл. с титул. экрана.

Представлены задания и краткие теоретические сведения к лабораторным работам для ознакомления с многоуровневой организацией вычислительных систем: логический уровень, уровни микроархитектуры, набора команд, операционной системы. Рассмотрены вопросы мониторинга и тестирования компонентов вычислительных систем.

Для студентов специальности 1-40 04 01 «Информатика и технологии программирования» дневной формы обучения.

УДК 004.42(075.8)
ББК 32.973.22я73

СОДЕРЖАНИЕ

Введение

Дисциплина «Архитектура вычислительных систем» является одной из базовых при подготовке студентов по специальности 1 - 40 04 01 - «Информатика и технологии программирования». Знания и умения, полученные при изучении дисциплины, необходимы для освоения последующих специальных дисциплин и позволит программистам рационально использовать все ресурсы вычислительной системы, а также проектировать эффективные программы..

Практикум по выполнению лабораторных работ ориентирован на формирование у студентов основных занний по принципам организации компьютерных систем, организации логического и микроархитектурного уровня, уровня набора команд и уровня операционной системы.

В практикуме представлены краткие теоретические сведения и задания к лабораторным работам. Выполнение лабораторных заданий предусмотренио как на основе низкоуровнего языка программирования ассемблера, так и на языках высокого уровня C++, C#. Отдельно рассматриваются вопросы моделирования и тестирования компонентов вычислительных систем, методов мониторинга.

Лабораторный практикум полностью соответствует учебной программе по дисциплине «Архитектруа вычислительных систем».

Лабораторная работа № 1
Представление данных. Системы счисления

Цель работы: ознакомиться с позиционными системами счисления, способами перевода данных из одной системы счисления в в другую, организацией арифметических операций над данными в позициионных системах.

Краткие теоретические сведения:

Основные понятия и определения

Система счисления – это способ представления чисел с помощью некоторого набора символов, называемых цифрами, а также совокупность приемов и правил, позволяющих однозначно представлять числовую информацию.

Все системы счисления можно разделить на позиционные и непозиционные.

В непозиционных системах счисления вес цифры (т. е. тот вклад, который она вносит в значение числа) не зависит от ее позиции в записи числа. Примером непозиционной системы счисления является римская система. В этой системе в качестве цифр используются латинские буквы:

I	V	X	L	C	D	M	и т.д. – римские цифры
1	5	10	50	100	500	1000	- десятичные эквиваленты римским цифрам.

Величина числа определяется суммой или разностью цифр в числе. Если меньшая цифра стоит слева от большей, то она вычитается, если справа — прибавляется. Например:

LXVI = L + X + V + I = 50 + 10 + 5 + 1 = 56

DCVL = D + C + (X – V) = 500 + 100 + (50–5) = 645

CDXLIX = (D - C) + (L-X) + (X - I) = (500-100) + (50-10) + (10-1) = 449

К недостаткам таких систем относятся наличие большого количества знаков и сложность выполнения арифметических операций.

В позиционных системах счисления вес каждой цифры изменяется в зависимости от ее положения (позиции) в последовательности цифр, изображающих число. Количественная оценка записанного числа в такой системе счисления определяется как сумма произведений значения цифр, составляющих запись числа, умноженных на вес позиции, в которой располагается цифра.

Примером позиционной системы счисления является широко используемая десятичная система счисления. Например, количественная оценка десятичного числа 777 определяется как

$7 \cdot 100 + 7 \cdot 10 + 7,$

где 100, 10, 1 - соответственно веса третьего, второго и первого разрядов записи оцениваемого числа.

Любая позиционная система счисления характеризуется своим основанием. Основание позиционной системы счисления — это количество различных цифр, используемых для изображения чисел в данной системе счисления. В десятичной системе используются десять цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Основанием этой системы является число десять.

За основание системы счисления можно принять любое натуральное число – два, три, четыре и т.д. Следовательно, возможно бесчисленное множество позиционных систем: двоичная, троичная, четверичная и т.д.

Для записи чисел в позиционной системе с основанием q необходимо иметь алфавит из n цифр. В качестве цифр используются обозначение соответствующих цифр десятичной системы счисления 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, а в случае, когда десятичных цифр «не хватает» (для систем счисления с основанием q, большим чем 10), для цифр, превышающих 9, вводятся дополнительные обозначения, например, для q =16 добавляют буквы *A, B, C, D, E, F*, которые соответствуют шестнадцатеричным цифрам, десятичные эквиваленты которых равны соответственно 10, 11, 12, 13, 14, 15.

Чтобы указать основание системы, к которой относится число, необходимо приписать нижний индекс к числу. Например:

$10101101_2, 1221_3, 747_8, 3BE5_{16}$

Запись чисел в каждой из систем счисления с основанием q означает сокращенную запись выражения

$$a_{n-1}\, q^{n-1} + a_{n-2}\, q^{n-2} + \ldots + a_1\, q^1 + a_0\, q^0 + a_{-1}\, q^{-1} + \ldots + a_{-m}\, q^{-m}, \qquad (1)$$

где a_i – цифры системы счисления; n и m – число целых и дробных разрядов, соответственно.

Такая форма записи числа называется развернутой или расширенной. Пользуясь этой формулой можно легко перевести число из системы счисления с любым основанием в десятеричную.

Пример:

$$\mathbf{101101_2} = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 =$$
$$= 1 \cdot 32 + 0 + 1 \cdot 8 + 1 \cdot 4 + 0 + 1 \cdot 1 = 45_{10};$$
$$\mathbf{703_8} = 7 \cdot 8^2 + 0 \cdot 8^1 + 3 \cdot 8^0 = 7 \cdot 64 + 0 + 3 \cdot 1 = 451_{10};$$
$$\mathbf{B2E4_{16}} = 11 \cdot 16^3 + 2 \cdot 16^2 + 14 \cdot 16^1 + 4 \cdot 16^0 =$$
$$= 11 \cdot 4096 + 2 \cdot 256 + 14 \cdot 6 + 4 \cdot 1 = 45796_{10}.$$

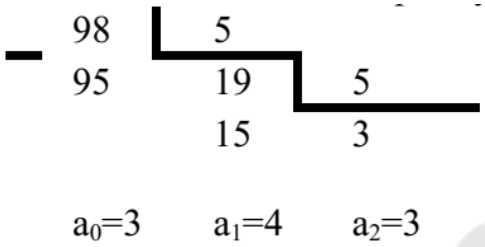
Перевод целых десятичных чисел в другие системы счисления

Перевод целых десятичных чисел в другие системы счисления осуществляется на основании следующего алгоритма:

- основание новой системы счисления необходимо выразить в десятичной системе счисления и все последующие действия производить в десятичной системе счисления;
- последовательно выполнять деление заданного числа и получаемых неполных частных на основание новой системы счисления до тех пор, пока не получится неполное частное, меньшее основания новой системы счисления;
- полученные остатки, являющиеся цифрами числа в новой системе счисления, необходимо привести в соответствие с алфавитом новой системы счисления;
- составить число в новой системе счисления, записывая его начиная с последнего остатка.

Пример:

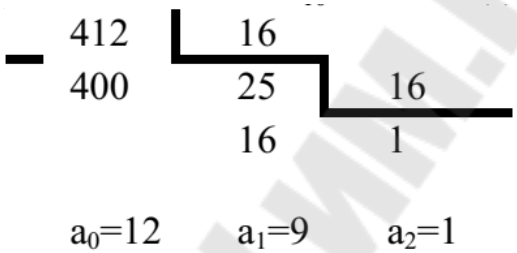
а) перевести число 98_{10} в пятеричную систему



$a_0=3$ $a_1=4$ $a_2=3$

Таким образом: $98_{10} = 343_5$

б) перевести число 412_{10} в шестнадцатеричную систему



$a_0=12$ $a_1=9$ $a_2=1$

Таким образом: $412_{10} = 19C_{16}$. Напомним, что $12_{10} = C_{16}$.

Системы счисления, используемые в цифровых компьютерах (с основанием 2^n)

В аппаратной основе вычислительных систем лежат двухпозиционные элементы, которые могут находиться только в двух состояниях; одно из них обозначается 0, а другое – 1. Поэтому основной системой счисления применяемой в цифровых компьютерах является двоичная система, которая обладает рядом преимуществ перед другими системами:

- для ее реализации нужны технические устройства с двумя устойчивыми состояниями (есть – ток нет тока, намагничен – не намагничен и т.п.), а не, например, с десятью, – как в десятичной;
- представление информации посредством только двух состояний надежно и помехоустойчиво;
- возможно применение аппарата булевой алгебры для выполнения логических преобразований информации;
- двоичная арифметика намного проще десятичной.

Недостаток двоичной системы — быстрый рост числа разрядов, необходимых для записи чисел.

Перевод чисел из десятичной системы в двоичную и наоборот выполняет вычислительная машина. Однако, чтобы профессионально использовать компьютер, следует научиться понимать слово машины. Для этого и разработаны восьмеричная и шестнадцатеричная системы. Числа в этих системах читаются почти так же легко, как десятичные, требуют соответственно в три (восьмеричная) и в четыре (шестнадцатеричная) раза меньше разрядов, чем в двоичной системе (ведь числа 8 и 16 — соответственно, третья и четвертая степени числа 2).

Для того чтобы целое двоичное число записать в системе с основанием $q=2^n$ (4, 8, 16 и т.д.), необходимо:

- заданное двоичное число разбить справа налево на группы по n цифр в каждой;
- если в последней левой группе окажется меньше n разрядов, то ее надо дополнить слева нулями до нужного числа разрядов;
- рассмотреть каждую группу как n-разрядное двоичное число и записать ее соответствующей цифрой в системе счисления с основанием $q=2^n$.

Для того чтобы произвольное число, записанное в системе счисления с основанием $q=2^n$, перевести в двоичную систему счисления, нужно каждую цифру этого числа заменить ее *n*-разрядным эквивалентом в двоичной системе счисления. Для перевода двоичного числа в восьмеричную и шестнадцатеричную системы счисления число разбивается на группы в 3 и 4 разряда соответственно. Для перевода удобно пользоваться двоично-восьмеричной (таблица 1.1) и двоично-шестнадцатеричной таблицами (таблица 1.2).

Таблица 1.1 – Двоично-восьмеричная

8	**2**
0	000
1	001
2	010

3	011
4	100
5	101
6	110
7	111

Таблица 1.2 – Двоично-шестнадцатеричная

16	**2**
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Пример:

1. Перевести число $5A2B_{16}$ двоичную систему.

Для перевода воспользуемся таблицей 1.2 и заменим каждую цифру в шестнадцатеричном числе на соответствующую ей в таблице четверку двоичных знаков. Получается:

0101 1010 0010 1011.

Лишние нули слева отбрасываются, так как они не влияют на значения целого числа. Таким образом:

$5A2B_{16} = 101101000101011_2$.

2. Перевести двоичное число $1101101110110010 1_2$ в шестнадцатеричную систему.

Разделим заданное число на группы по четыре цифры, начиная справа. Левую группу дополняем нулями.

0001 1011 0111 0110 0101

На основании данных из таблицы 1.2, заменяем каждую двоичную группу на соответствую шестнадцатеричную цифру (1 B 7 6 5). Следовательно: $1101101110110010 1_2 = 1B765_{16}$

3. Перевести двоичное число 1011101100101_2 в восьмеричную систему.

Разделим заданное число на группы по три цифры, начиная справа. Левую группу дополняем нулями.

001 011 101 100 101

На основании данных из таблицы 3, заменяем каждую двоичную группу на соответствующую восьмеричную цифру (1 3 5 4 5). Следовательно: $1011101100101_2 = 13545_8$.

Арифметические операции в позиционных системах счисления

Любая позиционная система счисления определяется основанием системы, алфавитом и правилами выполнения арифметических операций. Правила выполнения арифметических операций в десятичной системе счисления применимы и к другим позиционным системам счисления. Однако при выполнении операций сложения и умножения необходимо пользоваться таблицами сложения и умножения для конкретной системы счисления.

Например, правила выполнения операций сложения и умножения в пятеричной системе счисления задаются в виде таблицы 1.3.

Таблица 1.3 Сложение и умножение в пятеричной системе счисления

Сложение						Умножение					
+	**0**	**1**	**2**	**3**	**4**	*	**0**	**1**	**2**	**3**	**4**
0	0	1	2	3	4	**0**	0	0	0	0	0
1	1	2	3	4	10	**1**	0	1	2	3	4
2	2	3	4	10	11	**2**	0	2	4	11	13
3	3	4	10	11	12	**3**	0	3	11	14	22
4	4	10	11	12	13	**4**	0	4	13	22	31

Таблицы сложения и умножения легко составить, используя правило счета.

Выполняя умножение многозначных чисел в различных позиционных системах счисления, можно использовать обычный алгоритм перемножения чисел в столбик, но при этом результаты перемножения и сложения однозначных чисел необходимо заимствовать из соответствующих рассматриваемой системе таблиц умножения и сложения.

Деление в любой позиционной системе счисления производится по тем же правилам, как и деление углом в десятичной системе.

Задания к лабораторной работе:

Задание 1. Необходимо написать программу, которая на входе получает две строки из ASCII-символов. В первой строке указывается основание системы счисления, во второй число. На выходе должно выводиться десятичное число, соответствующее введенному. В программе предусмотреть проверку соответствия цифр введенного числа алфавиту заданной системы счисления.

Основание системы счисления выбирается согласно порядковому номеру студента в журнале + 3. Вводимое число соответствует четырем первым цифрам зачетной книжки.

Задание 2. Необходимо написать программу, которая на входе получает две строки из ASCII-символов. В первой строке задается десятичное число, во второй указывается основание системы счисления, в которую необходимо перевести заданное число. На выходе должно выводиться число в новой системе счисления в виде строки из ASCII-символов.

Основание системы счисления выбирается согласно порядковому номеру студента в журнале + 3. Вводимое число соответствует четырем первым цифрам зачетной книжки.

Задание 3.1. Необходимо написать программу, которая на входе получает строку из ASCII-символов, состоящую из 0 и 1, соответствующей записи заданного двоичного числа. На выходе должно выводиться 2 строки ASCII-символов, соответствующие шестнадцатеричному и двоичному представлению введенного числа. В программе предусмотреть проверку ввода некорректных данных.

Вводимое число соответствует четырем первым цифрам зачетной книжки, переведенное в двоичное число с помощью калькулятора.

Задание 3.2. Необходимо написать программу, которая на входе получает строку из ASCII-символов, соответствующей записи числа в заданной системе счисления (для нечетных вариантов – восьмеричная, для четных – шестнадцатеричная). На выходе должна выводиться строка ASCII-символов, состоящую из 0 и 1, соответствующая записи эквивалентного двоичного числа. В программе предусмотреть проверку ввода некорректных данных.

Вводимое число соответствует четырем первым цифрам зачетной книжки, переведенное в заданную систему счисления с помощью калькулятора.

Задание 4.1 Написать программу, осуществляющую сложение и вычитание чисел в заданной системе счисления. В программе предусмотреть проверку ввода некорректных данных.

Основание системы счисления выбирается согласно порядковому номеру студента в журнале + 3.

Задание 4.2. Написать программу, осуществляющую умножение и деление чисел в заданной системе счисления. В программе предусмотреть проверку ввода некорректных данных.

Основание системы счисления выбирается согласно порядковому номеру студента в журнале + 3.

Требование по содержанию отчета:

В отчете должны быть отображены следующие пункты:

1. Титульный лист.
2. Цель работы.
3. Задания.
4. Листинг программного кода с комментариями.
5. Тесты.
6. Результат выполнения программы.
7. Выводы

Контрольные вопросы для защиты:

1. Какое количество обозначает цифра 8 в десятичных числах 4587, 8652, 178, 894?
2. Выпишите алфавиты в 5-ричной, 7-ричной, 12-ричной системах счисления.
3. Запишите в развернутой форме числа: 3412_{10}, 3412_5, 3412_8, 3412_{12}.
4. Запишите в десятичной системе счисления числа: 421_5, 421_7, 421_9, 421_{16}.
5. Представить двоичные числа 1110101, 1001011, 10101011 в десятеричной системе счисления.

6. Представить восьмеричные числа 572, 765, 1274 в десятеричной системе счисления.
7. Представить десятичные числа 194 и 751 в пятеричной системе счисления.
8. Представить десятичные числа 241 и 967 в семеричной системе счисления.
9. Представить шестнадцатеричные числа 5A2, FE5, D2E41 в десятеричной системе счисления.
10. Перевести двоичные числа 10011101110, 1011011101 в восьмеричную систему счисления.
11. Перевести двоичные числа 10011101110, 1011011101 в шестнадцатеричную систему счисления.
12. Перевести двоичные числа 10110101110, 1011110101 в шестнадцатеричную систему счисления.
13. Перевести восьмеричные числа 472, 1435 в двоичную систему счисления.
14. Перевести шестнадцатеричные числа 4B7A, C43F в двоичную систему счисления.
15. Перевести шестнадцатеричные числа D719, 3EF6 в двоичную систему счисления.
16. Составьте таблицы сложения и умножения в троичной системе счисления и выполните вычисление 12 + 22, 212 - 12, 21 ·11, 130:2.
17. Вычислить выражение $10111_2 \times 110_2$.
18. Вычислить выражение $AFF1_{16} - 19D_{16}$.
19. В какой системе счисления выполнено умножение 213 х 3 = 1144?

Лабораторная работа № 2
Моделирование цифровых схем

Цель работы: получить навыки моделирования основных компонентов цифровых устройств в пакете моделирования электрических схем.

Краткие теоретические сведения:

Для моделирования и анализа электрических схем используются системы схемотехнического моделирования. Такие системы позволяют пользователю моделировать аналоговые, цифровые и цифро-аналоговые схемы большой степени сложности. Имеющиеся в системе библиотеки включают в себя большой набор широко распространенных электронных компонентов. Есть возможность подключения и создания новых библиотек компонентов. Параметры компонентов можно изменять в широком диапазоне значений. Простые компоненты описываются набором параметров, значения которых можно изменять непосредственно с клавиатуры, активные элементы − моделью, представляющей собой совокупность параметров и описывающей конкретный элемент или его идеальное представление. Модель выбирается из списка библиотек компонентов, параметры модели также могут быть изменены пользователем.

Для выполнения лбораторной работы можно использовать свободно распостраняемые программы Electronics Workbench или Logisim. Возможности Electronics Workbench гораздо шире, чем у Logisim, но для освоения наывоков моделирования и закрепления знаний по организации логического уровня компьютера достаточно любой из этих программ.

Electronics Workbench

Программа Electronics Workbench предназначена для моделирования и анализа электронных схем.

При создании схемы Electronics Workbench позволяет:

- выбирать элементы и приборы из библиотек;
- перемещать элементы и схемы в любое место рабочего поля;
- поворачивать элементы и их группы на углы, кратные 90 градусам;

– копировать, вставлять или удалять элементы, фрагменты схем;
– одновременно подключать несколько измерительных приборов и наблюдать их показания на экране монитора;
– присваивать элементам условные обозначения;
– изменять параметры элементов.

Изменяя настройки приборов можно:

– изменять шкалы приборов в зависимости от диапазона измерений;
– задавать режим работы прибора;
– задавать вид входных воздействий на схему (постоянные или гармонические токи или напряжения, треугольные или прямоугольные импульсы).

После запуска программы на экране появляются строка меню и панель компонентов (рисунок 1).

Панель компонентов состоит из пиктограмм полей компонентов, а поле компонентов - из условных изображений компонентов. Щелчком мыши на пиктограмме компонентов открывается поле соответствующее этой пиктограмме.

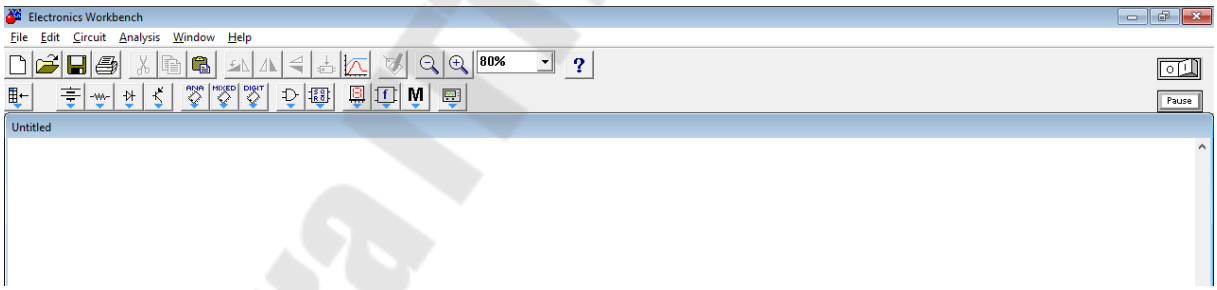
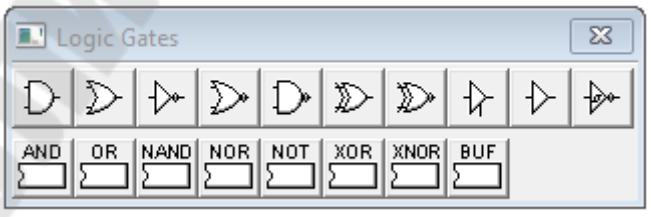


Рисунок 1 – Интерфейс программы

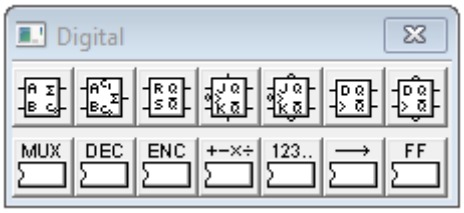
Ниже приведены некоторые элементы из полей компонентов, используемые в лабораторной работе:

Logic gates (основные вентили)



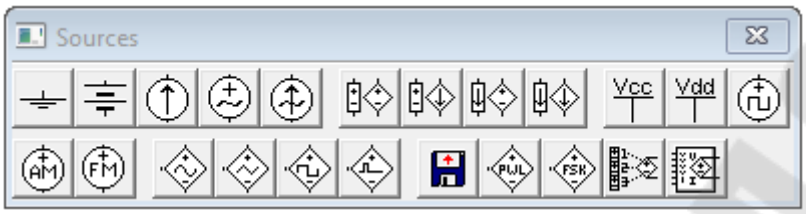
;

Digital (основные комбинаторные и арифметические схемы)



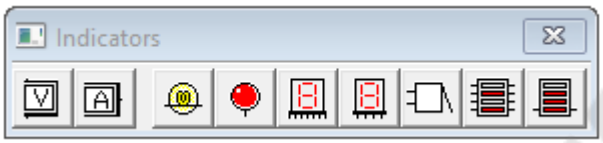
;

Sources (источники)



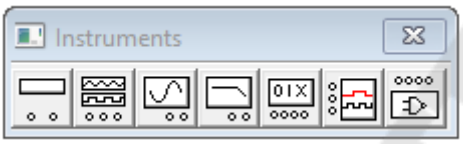
$

Indicators (приборы из библиотеки индикаторов)



;

Instruments



.

Моделирование схем

Исследуемая схема собирается на рабочем поле с использованием мыши и клавиатуры.

При построении и редактировании схем выполняются следующие операции:

- выбор компонента из библиотеки компонентов;
- выделение объекта;
- перемещение объекта;
- копирование объектов;
- удаление объектов;
- соединение компонентов схемы проводниками;
- установка значений компонентов;
- подключение измерительных приборов.

После построения схемы и подключения приборов анализ работы схемы начинается после нажатия выключателя в правом

верхнем углу окна программы (при этом в нижнем левом углу экрана показываются моменты схемного времени).

Повторное нажатие выключателя прекращает работу схемы.

Сделать паузу при работе схемы можно нажатием клавиши F9 на клавиатуре; повторное нажатие F9 возобновляет работу схемы (аналогичного результата можно добиться, нажимая кнопку Pause, расположенную под выключателем.)

Выбор компонента необходимого для построения схемы производится после выбора поля компонентов, содержащего необходимый элемент. Этот элемент захватывается мышью и перемещается на рабочее поле.

Выделение объекта. При выборе компонента необходимо щелкнуть на нем левой клавишей мыши. При этом компонент становится красным. (Снять выделение можно щелчком мыши в любой точке рабочего поля.)

Перемещение объекта. Для перемещения объекта его выделяют, устанавливают указатель мыши на объект и, держа нажатой левую клавишу мыши перетаскивают объект.

Объект можно поворачивать. Для этого объект нужно предварительно выделить, затем щелкнуть правой клавишей мыши и выбрать необходимую операцию:

- Rotate (поворот на 90 градусов);
- Flip vertical (переворот по вертикали);
- Flip horizontal (переворот по горизонтали).

Копирование объектов осуществляется командой Copy из меню Edit. Перед копированием объект нужно выделить. При выполнении команды выделенный объект копируется в буфер. Для вставки содержимого буфера на рабочее поле нужно выбрать команду Paste из меню Edit

Удаление объектов. Выделенные объекты можно удалить командой Delete.

Соединение компонентов схемы проводниками. Для соединения компонентов проводниками нужно подвести указатель мыши к выводу компонента (при этом на выводе появится черная точка). Нажав левую кнопку мыши, переместите ее указатель к выводу компонента, с которым нужно соединиться, и отпустите кнопку мыши. Выводы компонентов соединятся проводником.

Цвет проводника можно изменить, если дважды щелкнуть по проводнику мышью и выбрать из появившегося окна необходимый цвет.

Удаление проводника. Если по какой -либо причине проводник нужно удалить, необходимо подвести указатель мыши к выходу компонента (должна появиться черная точка). Нажав левую клавишу мыши, переместите ее на пустое место рабочего поля и отпустите кнопку мыши. Проводник исчезнет.

Установка значений параметров производится в диалоговом окне свойств компонента, которое открывается двойным щелчком мыши по изображению компонента (Закладка Value).

Каждому компоненту можно присвоить имя (Закладка Label)

Подключение приборов. Для подключения прибора к схеме нужно мышью перетащить прибор с панели инструментов на рабочее поле и подключить выводы прибора к исследуемым точкам. Некоторые приборы необходимо заземлять, иначе их показания будут неверными.

Пример 1: реализовать схему XOR (Исключающего-ИЛИ), используя только базовые вентили NOT, AND и OR.

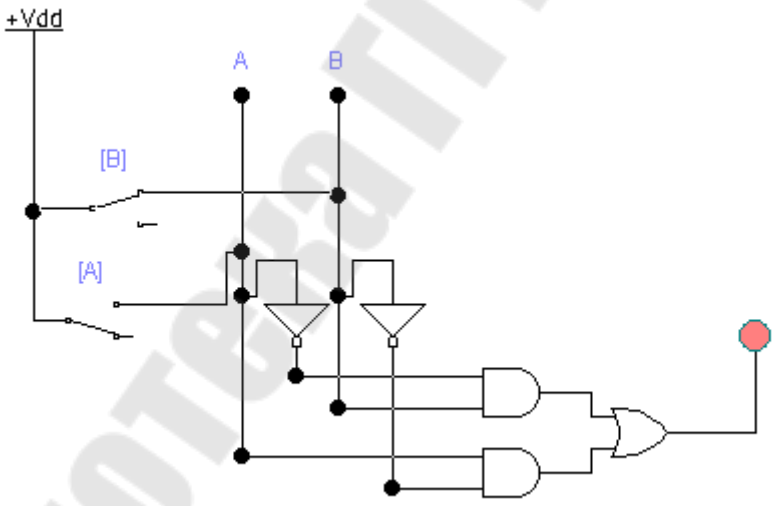


Рисунок 2 – Реализация схемы XOR

Пример 2: отобразить четырехразрядное двоичное число на индикаторе в виде десятичного.

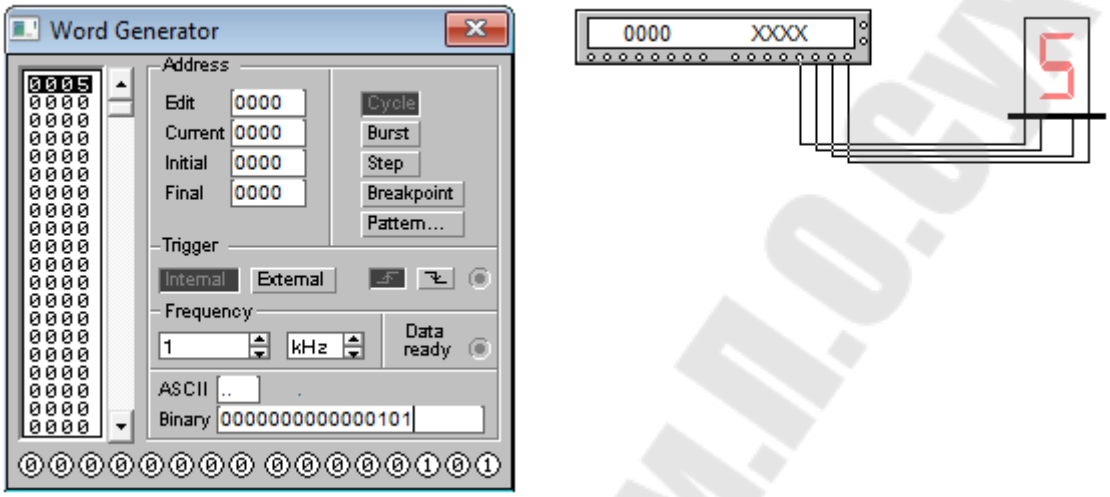


Рисунок 3 – Реализация с использование генератора слов

Задание к лабораторной работе:

1 Посчитать значение следующего выражения $S = \frac{K1+K2}{|K3-K4|} \times 10000$,

где К1- номер списка по журналу в десятичной системе счисления;

К2 – количество букв в фамилии в восьмеричной системе счисления;

К3 – год рождения в шестнадцатеричной системе счисления;

К4 – значение текущего года в двенадцатиричной системе счисления.

2 Перевести число S в двоичную систему с запятой и вставить вместо значений функции F (таблица 5), убрав лишние значащие разряды числа.

3 Составить функцию ДНФ.

4 Минимизировать полученную функцию любым из известных способов.

5 Реализовать логические функции (ДНФ и минимизированную) в пакете моделирования цифровых схем.

6 Проверить работоспособность разработанных схем при различных наборах входных сигналов.

Таблица 2.1

№	x1	x2	x3	x4	F(x1,x2,x3,x4)	S
1	0	0	0	0		Дробная часть
2	0	0	0	1		
3	0	0	1	0		
4	0	0	1	1		
5	0	1	0	0		
6	0	1	0	1		
7	0	1	1	0		
8	0	1	1	1		
9	1	0	0	0		Целая часть
10	1	0	0	1		
11	1	0	1	0		
12	1	0	1	1		
13	1	1	0	0		
14	1	1	0	1		
15	1	1	1	0		
16	1	1	1	1		

Требование по содержанию отчета:

В отчете должны быть отображены следующие пункты:

1. Титульный лист.
2. Цель работы.
3. Задание.
4. Вычисление логической функции.
5. Минимизация логической функции.
6. Реализация заданной и минимизированной логических функций в пакете моделирования цифровых схем.
7. Результаты работы модели с разным набором входных сигналов.
8. Выводы.

Контрольные вопросы для защиты:

1. Что такое логическая функция?
2. Как реализовать логическую функцию в пакете моделирования цифровых схем?
3. Основные правила алгебры логики.
4. Какие базовые вентили составляют основу цифровых схем?
5. Что такое таблица истинности?
6. Как перейти от таблицы истинности к описанию логическрй функции?
7. Что такое СДНФ?
8. Что такое СКНФ?
9. Какую логическую функцию выбрать для релизации для конкретной таблицы истинности СДНФ или СКНФ?
10. Как минимизировать логическую функцию?
11. Методы минимизации логических функций.

Лабораторная работа № 3
Моделирование комбинаторных и арифметических схем

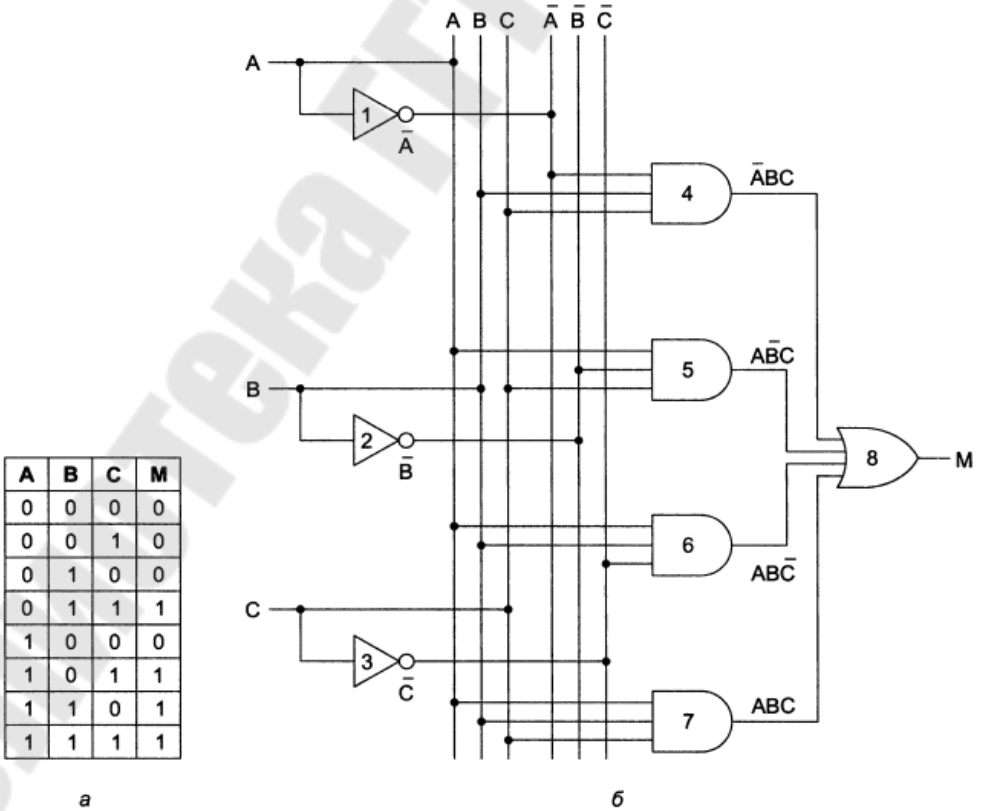
Цель работы: научиться моделировать комбинаторные и арифметические схемы, а также ознакомиться с реально существующими микросхемами на основе библиотеки встроенных интегральных схем.

Краткие теоретические сведения:

Комбинаторные схемы

Многие применения цифровой логики требуют наличия схем с несколькими входами и несколькими выходами, в которых выходные сигналы определяются текущими входными сигналами. Такая схема называется комбинаторной.

Микросхема, которая реализует таблицу истинности (рисунок 3.1), является типичным примером комбинаторной схемы.



A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

а *б*

Рисунок 3.1 – Комбинаторная схема большинства из 3

Рассмотрим наиболее часто используемые комбинаторные схемы.

Мультиплексоры

На цифровом логическом уровне **мультиплексор** представляет собой схему с 2^n входами, одним выходом и n линиями управления, которые позволяют выбрать один из входов. Выбранный вход соединяется с выходом. На рисунке 3.2 изображена схема восьмивходового мультиплексора. Три линии управления, A, B и C, кодируют 3-разрядное число, которое указывает, какая из восьми входных линий должна соединяться с вентилем ИЛИ и, следовательно, с выходом. Вне зависимости от того, какое значение окажется на линиях управления, семь вентилей И всегда будут выдавать на выходе 0, а оставшийся может выдавать 0 или 1 в зависимости от значения выбранной линии входа. Каждый вентиль И запускается определенной комбинацией сигналов на линиях управления. Если в схему мультиплексора, показанную на рисунке 3.2, добавить источник питания и землю, то мультиплексор можно включить в корпус с 14 выводами.

Использовав мультиплексор, мы можем реализовать функцию большинства (см. рисунок 3.1, а), как показано на рисунке 3.3, б. Для каждой комбинации A, B и C выбирается одна из входных линий. Каждый вход соединяется либо с сигналом V_{cc} (логическая 1), либо с землей (логический 0). Алгоритм соединения входов очень прост: входной сигнал D_i такой же, как значение в строке i таблицы истинности. На рисунке 3.1, а в строках 0, 1, 2 и 4 значение функции равно 0, поэтому соответствующие входы заземляются; в оставшихся строках значение функции равно 1, поэтому соответствующие входы соединяются с логической единицей.

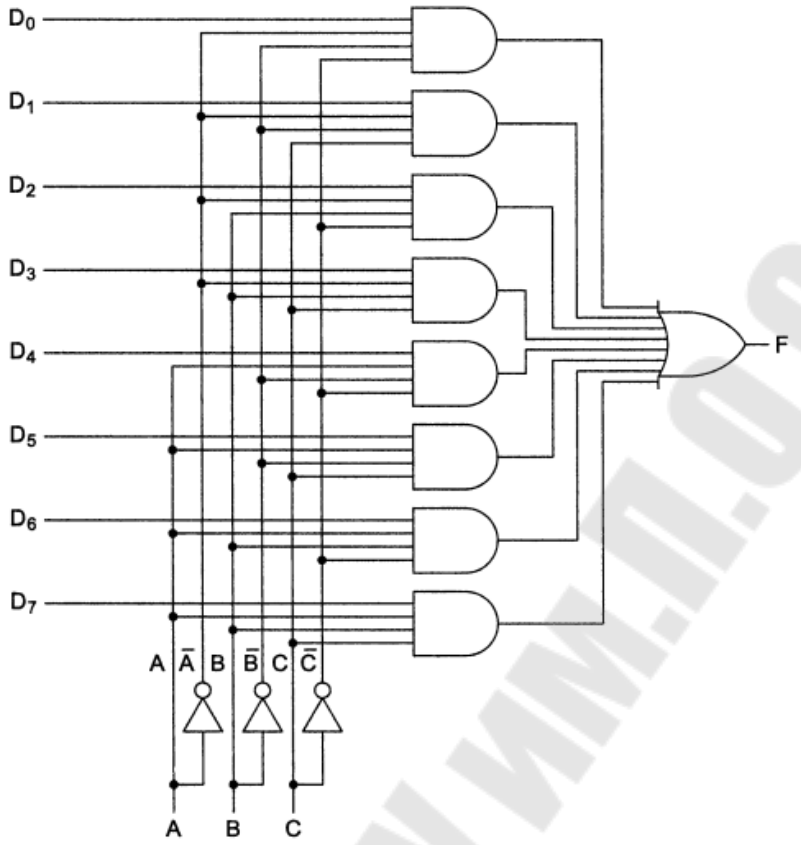


Рисунок 3.2 – Схема восьмивходового мультиплексора

Таким способом можно реализовать любую таблицу истинности с тремя переменными, использовав микросхему на рисунке 3.3, а.

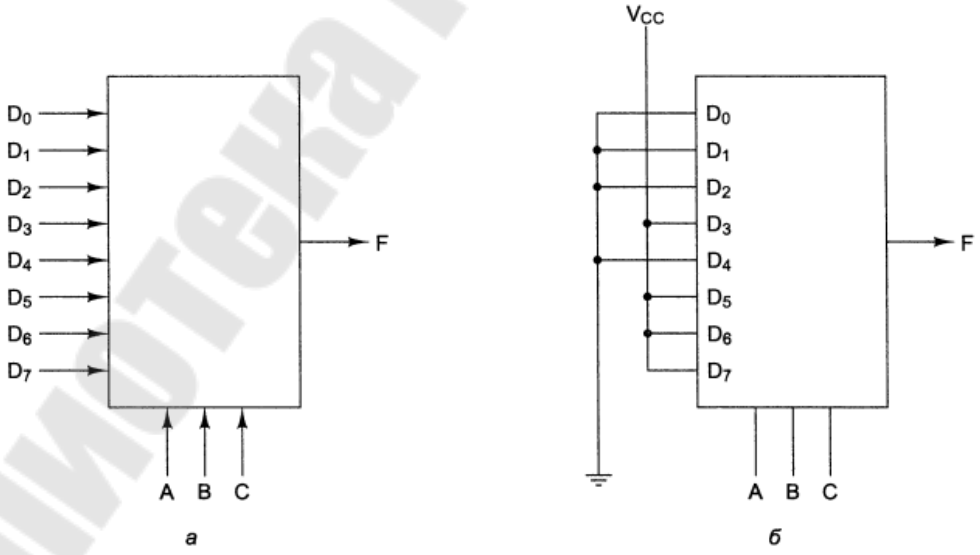


Рисунок 3.3 - Мультиплексор, построенный на СИС (а); тот же мультиплексор, смонтированный для вычисления функции большинства (б).

Противоположностью мультиплексора является **демультиплексор**, который соединяет единственный входной сигнал с одним из 2^n выходов в зависимости от значений сигналов в n линиях управления. Если бинарное значение линий управления равно k, то выбирается выход k.

Декодеры

Рассмотрим схему, которая получает на входе n-разрядное число и использует его для того, чтобы выбрать (то есть установить в значение 1) одну из 2^n выходных линий. Такая схема называется **декодером**.

Пример декодера для n = 3 показан на рисунке 3.4.

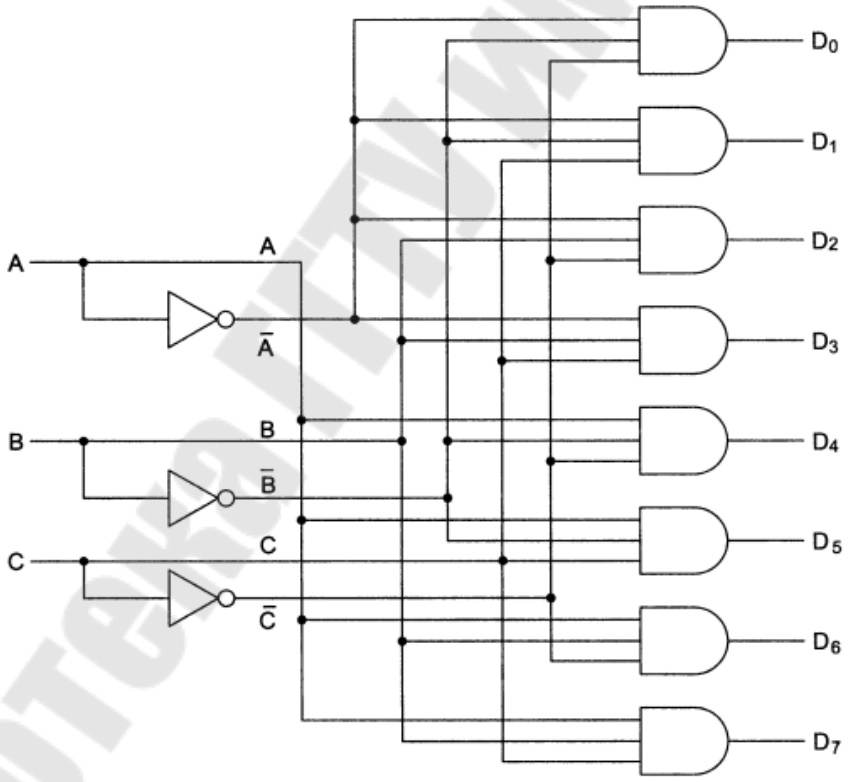


Рисунок 3.4 – Схема декодера, содержащего 3 входа и 8 выходов

Чтобы понять, зачем нужен декодер, представим себе память, состоящую из 8 микросхем, каждая из которых содержит 256 Мбайт. Микросхема 0 имеет адреса от 0 до 255 Мбайт, микросхема 1 — адреса от 256 Мбайт до 511 Мбайт и т. д. Три старших двоичных

разряда адреса используются для выбора одной из восьми микросхем. На рисунке 6.7 эти три бита — три входа A, B и C. В зависимости от входных сигналов ровно одна из восьми выходных линий (D0, ..., D7) принимает значение 1; остальные линии принимают значение 0. Каждая выходная линия запускает одну из восьми микросхем памяти. Поскольку только одна линия принимает значение 1, запускается только одна микросхема.

Компараторы

Компаратор сравнивает два слова, которые поступают на вход. Компаратор, изображенный на рисунке 3.5, принимает два входных сигнала, A и B по 4 бита каждый и выдает 1, если они равны, и 0, если они не равны. Схема основывается на вентиле ИСКЛЮЧАЮЩЕЕ ИЛИ, который выдает 0, если сигналы на входе равны, и 1, если сигналы на входе не равны. Если все четыре входных слова равны, все четыре вентиля ИСКЛЮЧАЮЩЕЕ ИЛИ должны выдавать 0. Эти четыре сигнала затем поступают в вентиль ИЛИ. Если в результате получается 0, значит, слова, поступившие на вход, равны; в противном случае они не равны. В нашем примере мы использовали вентиль ИЛИ-НЕ в качестве конечного, чтобы поменять значение полученного результата: 1 - означает равенство, 0 - неравенство.

Арифметические схемы

Перейдем от схем общего назначения к комбинаторным схема, которые используются для выполнения арифметических операций.

Схемы сдвига

Первой арифметической схемой, которую мы рассмотрим, будет схема сдвига, содержащая 8 входов и 8 выходов (рисунок 3.6). Восемь входных битов подаются на линии D0, … , D7. Выходные данные, которые представляют собой входные данные, сдвинутые на 1 бит,

поступают на линии S0, ... , S7. Линия управления C определяет направление сдвига: 0 — влево, 1 — вправо. При сдвиге влево в бит 7 вставляется 0. Аналогичным образом при сдвиге вправо в бит 0 вставляется значение 1.

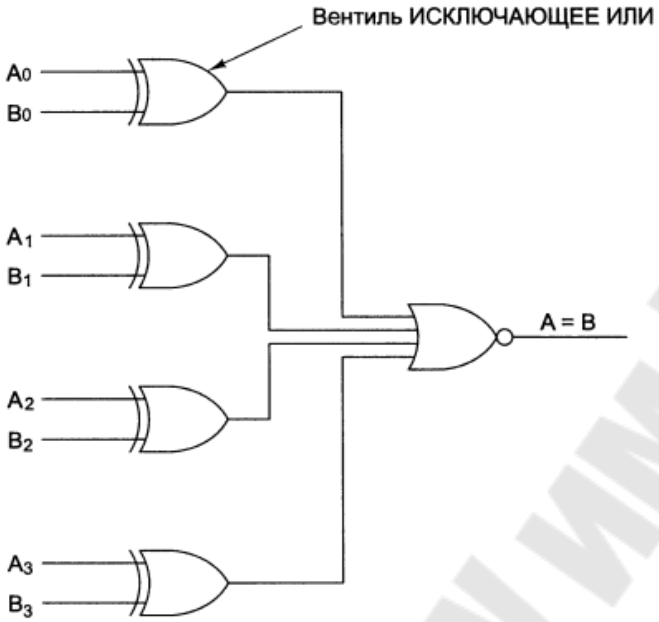


Рисунок 3.5 – Простой 4-разрядный компаратор

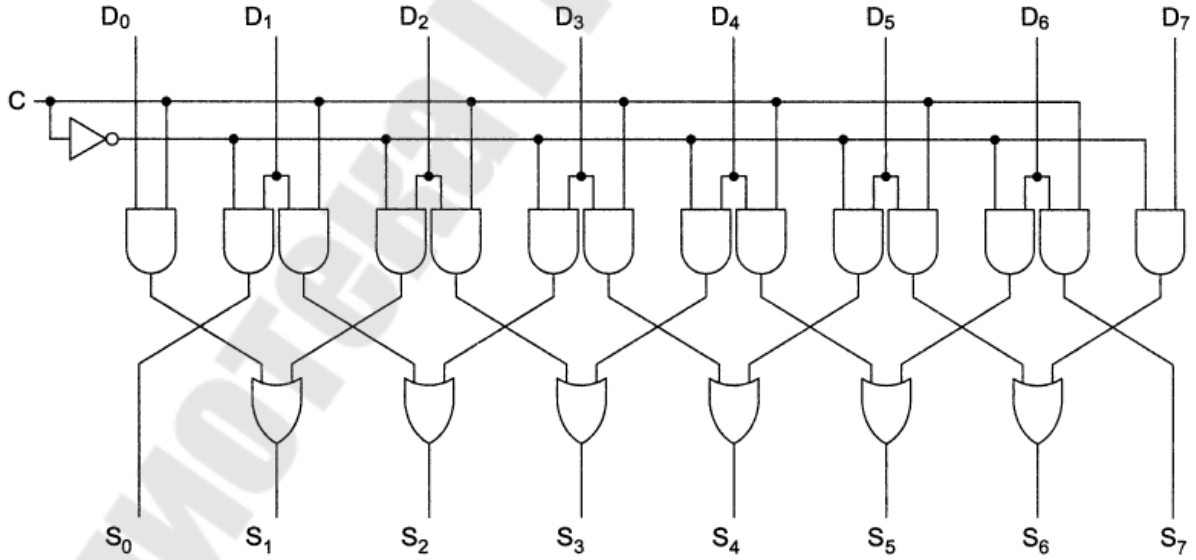


Рисунок 3.6 – Схема сдвига

Чтобы понять, как работает такая схема, рассмотрим пары вентилей И (кроме крайних). Если С=1, правый член каждой пары включается, пропуская через себя соответствующий бит. Так как правый вентиль И соединен с входом вентиля ИЛИ, который расположен справа от этого вентиля И, происходит сдвиг вправо. Если С=0, включается левый вентиль И из пары, и тогда происходит сдвиг влево.

Сумматоры

Схема выполнения операций сложения является существенной частью любого процессора. Таблица истинности для сложения одноразрядных целых чисел показана на рисунке 3.7, а. Здесь имеется два результата: сумма входных переменных А и В и перенос на следующую (левую) позицию. Схема для вычисления бита суммы и бита переноса показана на рисунке 3.7, б. Такая схема обычно называется **полусумматором**.

A	B	Сумма	Перенос
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

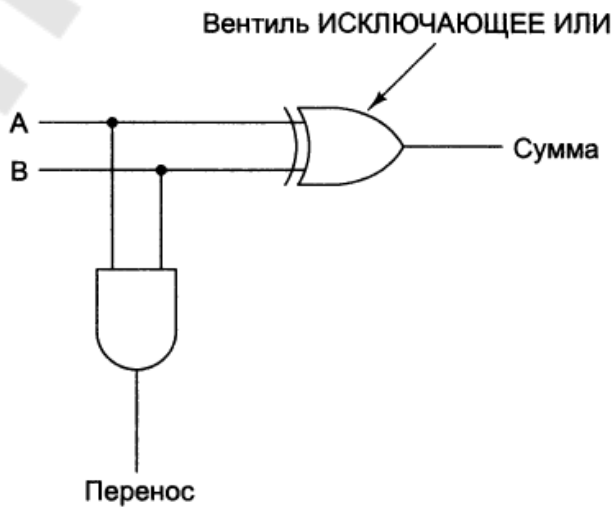


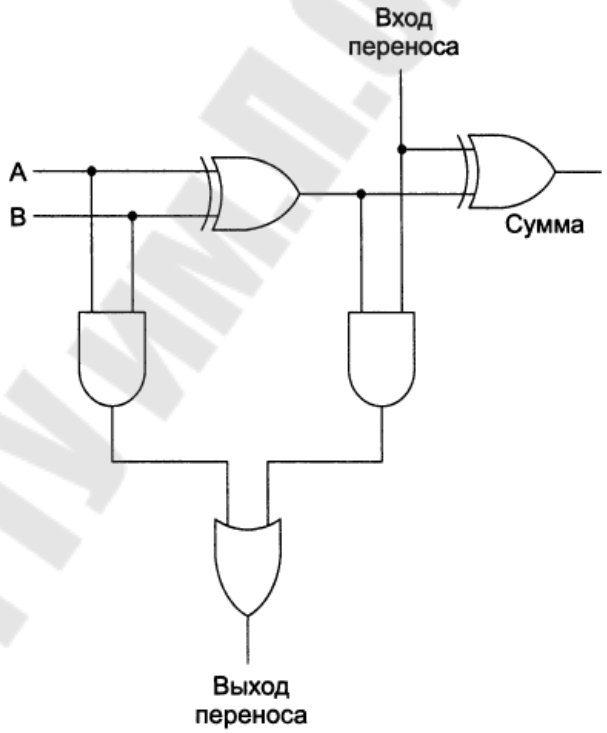
Рисунок 3.7 – Таблица истинности для сложения одноразрядных чисел (а); схема полусумматора (б)

Полусумматор подходит для сложения битов нижних разрядов двух многобитовых слов. Однако он не годится для сложений битов в середине слова, потому что не может осуществлять перенос в эту

позицию. Поэтому необходим полный сумматор (рисунок 3.8). Из схемы должно быть ясно, что полный сумматор состоит из двух полусумматоров. Сумма равна 1, если нечетное число переменных A, B и вход переноса принимает значение 1 (то есть, если единице равна или одна из переменных, или все три). Выход переноса принимает значение 1, если либо A и B одновременно равны 1 (левый вход в вентиль ИЛИ), либо один из них равен 1 и вход переноса также равен 1. Два полусумматора порождают и биты суммы, и биты переноса.

A	B	Вход переноса	Сумма	Выход переноса
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

а



б

Рисунок 3.8 – Таблица истинности для полного сумматора (а); схема для полного сумматора (б)

Чтобы построить сумматор, например, для двух 16-разрядных слов, нужно 16 раз продублировать схему, изображенную на рисунке 6.12, б. Перенос производится в левый соседний бит. Перенос в самый правый бит соединен с 0. Такой сумматор называется **сумматором со сквозным переносом**. Прибавление 1 к числу 111...111 не осуществится до тех пор, пока перенос не пройдет весь путь от самого правого бита к самому левому. Существуют более быстрые

сумматоры, работающие без подобной задержки. Естественно, предпочтение обычно отдается им.

Рассмотрим пример более быстрого сумматора. Разобьем 32-разрядный сумматор на 2 половины: нижнюю 16-разрядную и верхнюю 16-разрядную. Когда начинается сложение, верхний сумматор еще не может приступить к работе, поскольку не знает значение переноса, а узнать его он не сможет, пока не совершится 16 суммирований в нижнем сумматоре.

Однако можно сделать одно преобразование. Вместо одного верхнего сумматора можно получить два верхних сумматора, продублировав соответствующую часть аппаратуры. Тогда схема будет состоять из трех 16-разрядных сумматоров: одного нижнего и двух верхних, U0 и U1 работающих параллельно. В качестве переноса в сумматор U0 поступает 0, в сумматор U1 — 1. Оба верхних сумматора начинают работать одновременно с нижним сумматором, но только один из результатов суммирования в двух верхних сумматорах будет правильным. После сложения 16 нижних разрядов становится известно значение переноса в верхний сумматор, и тогда можно определить правильный ответ. При подобном подходе время сложения сокращается в два раза. Такой сумматор называется **сумматором с выбором переноса**. Можно еще раз разбить каждый 16-разрядный сумматор на два 8-разрядных и т. д.

Арифметико-логические устройства

Большинство компьютеров содержат одну схему для выполнения над двумя машинными словами операций И, ИЛИ и сложения. Обычно эта схема для n-разрядных слов состоит из n идентичных схем — по одной для каждой битовой позиции. На рисунке 3.9 изображена такая схема, которая называется **арифметико-логическим устройством (АЛУ)**. Это устройство может вычислять одну из 4-х следующих функций: A И B, A ИЛИ B, $\bar{B}$ или A + B. Выбор функции зависит от того, какие сигналы поступают на линии F0 и F1: 00, 01, 10 или 11 (в двоичной системе

счисления). Отметим, что здесь A + B означает арифметическую сумму A и B, а не логическую операцию ИЛИ.

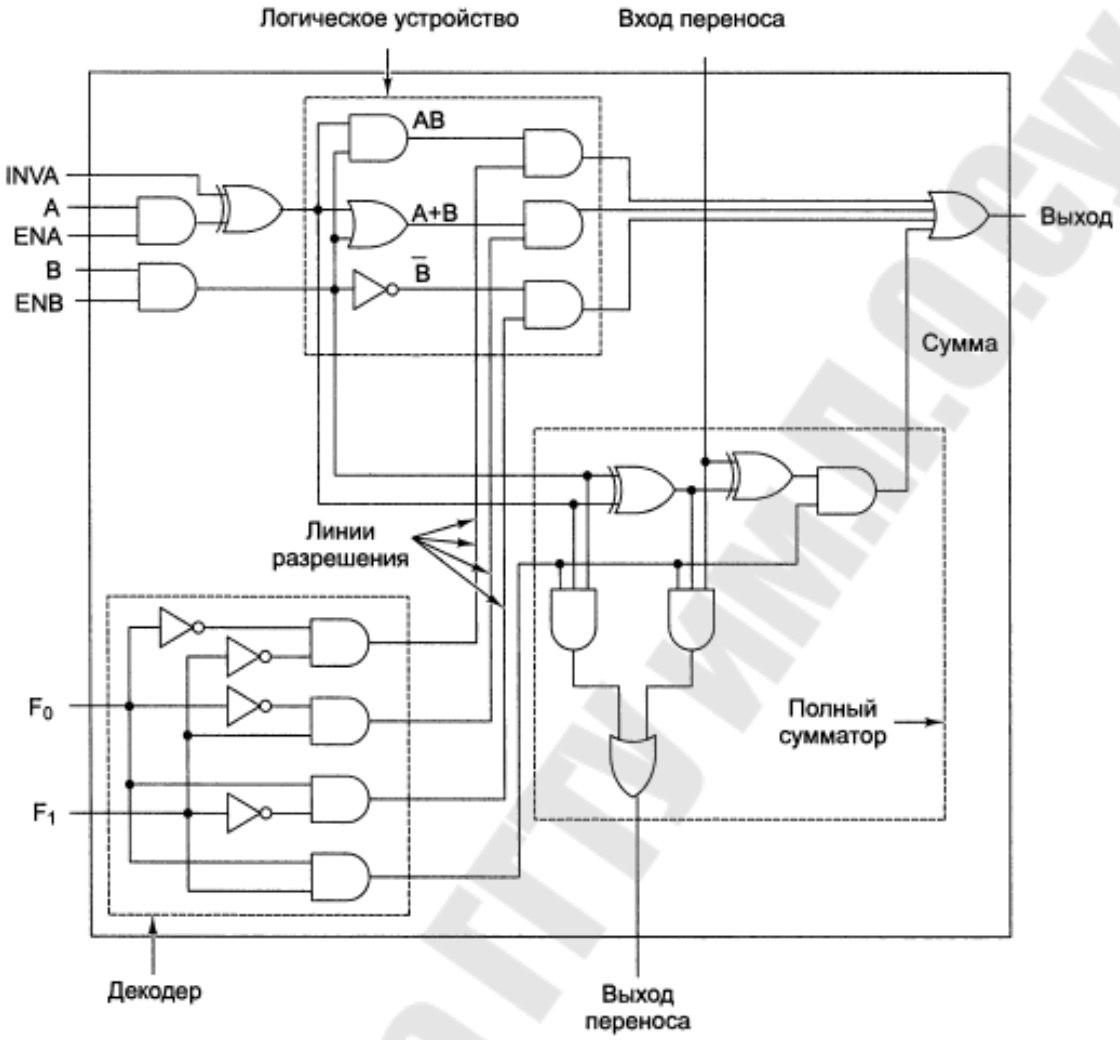


Рисунок 3.9 – Одноразрядное АЛУ

В левом нижнем углу схемы находится двухразрядный декодер, который генерирует сигналы включения для четырех операций. Выбор операции определяется сигналами управления F0 и F1. В зависимости от значений F0 и F1 выбирается одна из четырех линий разрешения, и тогда выходной сигнал выбранной функции проходит через последний вентиль ИЛИ.

В верхнем левом углу схемы находится логическое устройство для вычисления функций A И B, A ИЛИ B и $\overline{B}$, но только один из этих результатов проходит через последний вентиль ИЛИ в зависимости от того, какую из линий разрешения выбрал декодер. Так

как ровно один из выходных сигналов декодера может быть равен 1, то и запускаться будет ровно один из четырех вентилей И. Остальные три вентиля будут выдавать 0 независимо от значений A и B.

АЛУ может выполнять не только логические и арифметические операции над переменными A и B, но и делать их равными нулю, отрицая ENA (сигнал разрешения A) или ENB (сигнал разрешения B). Можно также получить $\overline{A}$, установив сигнал INVA (инверсия A). При нормальных условиях и ENA, и ENB равны 1, чтобы разрешить поступление обоих входных сигналов, а сигнал INVA равен 0. В этом случае A и B просто поступают в логическое устройство без изменений.

В нижнем правом углу находится полный сумматор для подсчета суммы A и B, а также для осуществления переносов. Переносы необходимы, поскольку несколько таких схем могут быть соединены для выполнения операций над целыми словами. Одноразрядные схемы, подобные показанной на рисунке 6.13, называются разрядными микропроцессорными секциями. Они позволяют разработчику строить АЛУ любой разрядности. На рисунке 3.10 показана схема 8-разрядного АЛУ, составленного из восьми одноразрядных секций. Сигнал INC (увеличение на единицу) нужен только для операций сложения. Он дает возможность вычислять такие суммы, как A+1 и A+B+1.

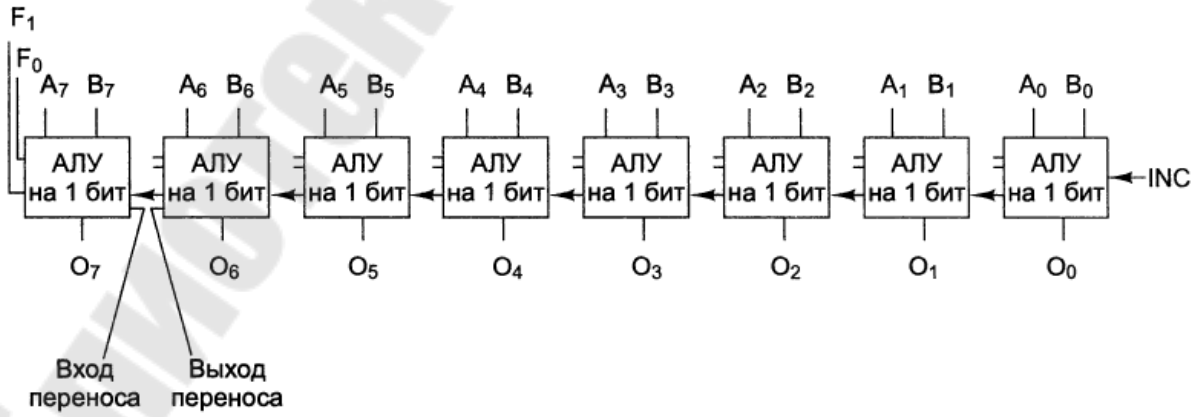


Рисунок 3.10 – Восемь одноразрядных секций, соединенных в 8-разрядное АЛУ. Для упрощения схемы сигналы разрешения и инверсии не показаны

Задание к лабораторной работе:

1. В пакете моделирование реализовать комбинационную схему функции большинства, разобранную (рисунок 3.1).
2. Реализовать с использованием библиотечных мультиплексоров логическую функцию своему варианту (функция задается в лабораторной работе №2).
3. Разработать полный сумматор для 3-х разрядных слов.
4. Разработать схему сдвига для 4-х разрядных слов: четные варианты – сдвиг влево, нечетные сдвиг – вправо.
5. Разработать схему одноразрядного АЛУ (арифметико-логического устройства), выполняющего следующие операции: инверсия входных сигналов; логическое ИЛИ входных сигналов, логическое И входных сигналов, арифметическая сумма входных сигналов с учетом знака переноса.

Требование по содержанию отчета:

Отчет должен включать следующие пункты:

1. Задание.
2. Реализация заданных схем в пакете моделирования (каждая схема – отдельная модель).
3. Результаты работы реализованных схем на различных наборах входных сигналов.
4. Выводы.

Контрольные вопросы для защиты:

1. Какую функцию выполняет мультиплексор?
2. Какую функцию выполняет демультиплексор?
3. Какую функцию выполняет декодер (дешифратор)?
4. Какую функцию выполняет шифратор?
5. Какую функцию выполняет компаратор?
6. Какую функцию выполняет сумматор?
7. Как реализовать компаратор на «больше» или «меньше»?
8. Какие основные операции реализуются в АЛУ?
9. Как нарастить разрядность арифметико-логического устройства?

Лабораторная работа № 4
Машинные команды и режимы адресации

Цель работы: изучить режимы адресации и форматы машинных команд.

Краткие теоретические сведения:

Структура машинной команды

Типовая команда, в общем случае, должна указывать:

- подлежащую выполнению операцию;
- адреса исходных данных (операндов), над которыми выполняется операция;
- адрес, по которому должен быть помещен результат операции.

В соответствии с этим команда состоит из двух частей: операционной и адресной (рисунок 4.1).

Операционная часть	Адресная часть

Рисунок 4.1 – Структура команды

Каждому действию, принадлежащему системе команд процессора, ставится во взаимно однозначное соответствие двоичный код, который принято называть кодом операции (КОП). Данные (число, логическое значение, символ, строка, бит, адрес поля памяти), участвующие в выполнении действия, обычно называются операндами. Операнд может быть задан непосредственно в команде, он может находиться в регистре процессора или в поле оперативной памяти. В любом случае операнд представлен некоторым двоичным кодом. Результат может быть помещен на хранение в один из регистров процессора или же в поле оперативной памяти. Если для операнда или результата используется регистр процессора, то необходимо указать, какой именно. В машинной команде это делается с помощью задания соответствующего регистру двоичного кода. Если используется поле памяти, необходимо указать его адрес, который также представляет собой двоичный код.

Машинная команда в целом представляет собой некоторый двоичный код, который может быть помещен в регистр процессора или в поле оперативной памяти. Для сокращения записи машинные команды обычно задаются в шестнадцатеричном виде.

Формат команды определяет ее структуру, то есть количество двоичных разрядов, отводимых под всю команду, а также количество и расположение отдельных полей команды. Полем называется совокупность двоичных разрядов, кодирующих составную часть команды.

В рамках системы команд одной вычислительной машины (ВМ) могут использоваться разные форматы команд. Обычно это связано с применением различных способов адресации. В таком случае в состав кода команды вводится поле для задания способа адресации (СА), и обобщенный формат команды приобретает вид, показанный на рисунке 4.2.

Рисунок 4.2 – Обобщенный формат команды

В большинстве ВМ одновременно используются несколько различных форматов команд.

Количество адресов в команде

Для определения количества адресов, включаемых в адресную часть, используется термин адресность. В «максимальном» варианте необходимо указать три компонента: адрес первого операнда, адрес второго операнда и адрес ячейки, куда заносится результат операции. В принципе может быть добавлен еще один адрес, указывающий место хранения следующей инструкции. В итоге имеет место четырехадресный формат команды (рисунок 4.3). Такой формат поддерживался в ВМ EDVAC, разработанной в 1940-х годах.

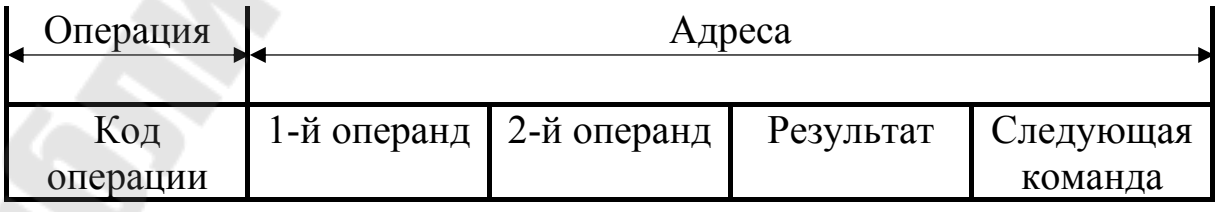


Рисунок 4.3 – Четырехадресный формат команды

В фон-неймановских ВМ необходимость в четвертом адресе отпадает, поскольку команды располагаются в памяти в порядке их выполнения, и адрес очередной команды может быть получен засчет простого увеличения адреса текущей команды в счетчике команд. Это позволяет перейти к трехадресному формату команды (рисунок 4.4). Требуется только добавить в систему команд ВМ команды, способные изменять порядок вычислений.

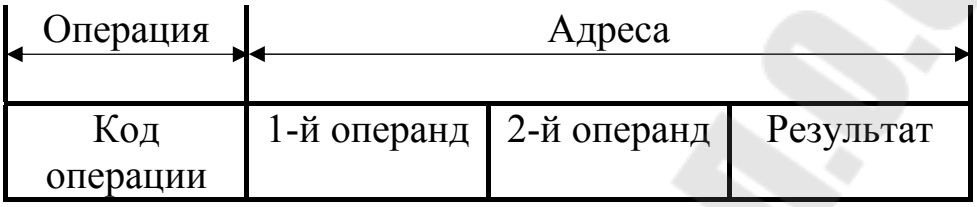


Рисунок 4.4 – Трехадресный формат команды

К сожалению, и в трехадресном формате длина команды может оказаться весьма большой. Так, если адрес ячейки основной памяти имеет длину 32 бита, а длина кода операции – 8 бит, то длина команды составит 104 бита (13 байт).

Если по умолчанию взять в качестве адреса результата адрес одного из операндов (обычно второго) то можно обойтись без третьего адреса, и в итоге получаем двухадресный формат команды (рисунок 4.5). Естественно, что в этом случае соответствующий операнд после выполнения операции теряется.

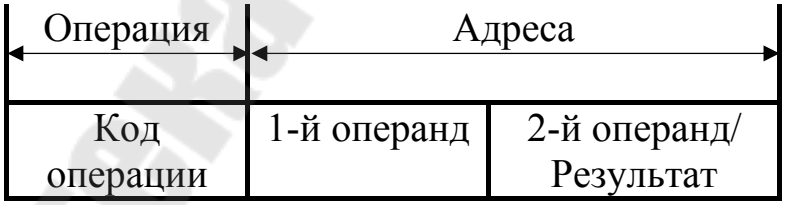


Рисунок 4.5 – Двухадресный формат команды

Команду можно еще более сократить, перейдя к одноадресному формату (рисунок 4.6), что возможно при выделении определенного стандартного места для хранения первого операнда и результата. Обычно для этой цели используется специальный регистр центрального процессора (ЦП), известный под названием аккумулятора, поскольку здесь аккумулируется результат.

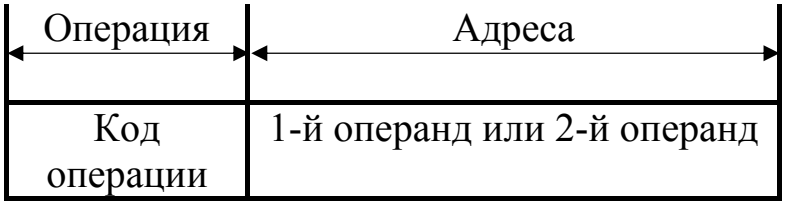


Рисунок 4.6 – Одноадресный формат команды

Наконец, если для обоих операндов указать четко заданное местоположение, а также в случае команд, не требующих операнда, можно получить нульадресный (безадресный) формат команды (рисунок 4.7). В таком варианте адресная часть команды вообще отсутствует или незадействуется.

Рисунок 4.7 – Нульадресный (безадресный) формат команды

Система команд процессора i8086 смешанная, она содержит нульадресные (безадресные), одно и двухадресные команды. При этом в двухадресных командах оба операнда не могут одновременно находиться в оперативной памяти. Один из них должен быть задан непосредственно либо выбираться из регистра процессора. Невозможны также команды, которые содержат оба операнда непосредственно в команде.

Способы адресации

Способы задания операндов в команде принято называть адресацией. Существует несколько десятков различных способов адресации, но, в принципе, все эти способы являются различными вариантами четырех основных:

- **непосредственная адресация** означает, что сам операнд (точнее, его код) включается в машинную команду как ее составная часть;
- **регистровая адресация** означает, что операнд находится в регистре процессора, а в команде указывается код этого регистра;
- **прямая адресация** означает, что операнд находится в поле оперативной памяти, а в команде указан адрес этого поля;
- **косвенная адресация** означает, что операнд также находится в поле оперативной памяти, а в команде содержатся некоторые элементы, по которым однозначно определяется адрес этого поля.

Непосредственная адресация при всей ее простоте имеет ограниченные возможности применения, так как при любом изменении операнда приходится изменять и содержащую его код машинную команду. Это означает изменение и программы в целом, что весьма нежелательно, так как в большинстве случаев требует ее повторной подготовки к выполнению (например, повторной трансляции). Непосредственно в машинной команде могут быть заданы только такие операнды, которые не меняются при любых выполнениях программы.

Использование для задания операндов регистровой адресации является самым эффективным и самым простым способом. Однако, как мы знаем, процессор i8086 обладает малым количеством регистров. Поэтому неизбежны сложности при организации вычислений с большим количеством различных данных. Кроме того, в начале выполнения программы все ее данные размещаются в оперативной памяти. Следовательно, в любом случае необходимы способы адресации, обеспечивающие возможность выборки операндов из полей оперативной памяти. Такими возможностями обладают прямая и косвенная адресация. В прямой адресации адрес поля памяти прямо указывается в команде, а в косвенной его необходимо определить по заданной в команде информации. Косвенная адресация по сравнению с прямой обеспечивает большую гибкость, необходимую при работе с данными сложной структуры, такими как массивы и записи.

Машинный и ассемблерный форматы команд

В качестве примера машинной команды рассмотрим одноадресную команду с кодом 0100 010122 или 4516. Первые пять битов этой команды, 010002, являются кодом операции, а последние три, 1012, - кодом регистра процессора bp. По этой команде осуществляются следующие действия:

- из регистра bp выбирается двухбайтный код и переносится в АЛУ;
- к выбранному коду прибавляется 1;
- результат записывается назад в регистр bp.

Таким образом, машинная команда увеличивает на единицу число, код которого находится в регистре bp. Это действие называется инкрементом операнда.

Запись команды в двоичном или шестнадцатеричном виде называется машинным форматом команды. Человеку записывать и воспринимать команды в таком виде очень сложно. Это требует значительных усилий по запоминанию машинных кодов, особенностей задания адресов, деталей выполнения действий и т. д. Поэтому разработан специальный способ записи машинных команд в символьном виде, который намного удобнее для человека. Вместо шестнадцатеричных или двоичных кодов в этом виде используются так называемые мнемокоды, то есть коды, удобные для запоминания. Мнемокоды представляют собой специально подобранные буквенно-цифровые обозначения, целые слова или сокращения какого-либо естественного языка, например, английского или русского. Запись команд с использованием мнемокодов называется ассемблерным форматом.

Таблица 4.1 – Машинный и ассемблерный формат команд

Машинный формат		**Ассемблерный формат**	**Действие**
Двоичный	**Шестнадцатеричный**		
$0100\ 0101_2$	45_{16}	inc bp	bp = bp + 1

В системе команд процессора i8086 предусмотрены два варианта команд: **регистровая** – более простая, но с ограниченными возможностями, – и **общая модификация**.

На рисунке 4.8 представлены две формы машинной команды с непосредственным операндом.

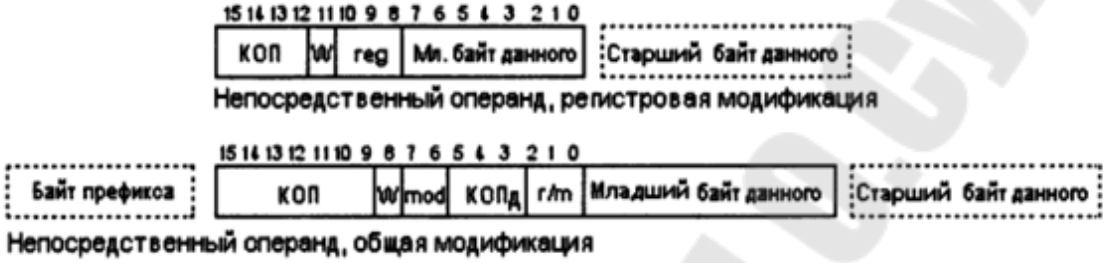


Рисунок 4.8 – Две формы машинной команды с непосредственным операндом

Обозначения имеют следующий смысл:

- КОП (7 битов) — код операции (основная часть кода);
- КОПД (3 бита) — дополнительный код операции (уточняющая часть кода);
- W (1 бит) — признак длины операнда. 0 — операнд однобайтовый, 1 — операнд двухбайтовый;
- mod (от modification — модификатор, 2 бита) — параметр, определяющий местоположение операнда;
- r/m (от register/memory — регистр/память, 3 бита) — параметр, уточняющий адресацию операнда.

Код операции в командах этого формата разбит на две части, КОП и КОПД, которые занимают соответственно разряды с 9-го no 15-й и с 0-го по 2-й двух основных байтов команды. Находящийся в 8-м разряде признак W определяет длину операнда команды. Если W — 0, то команда работает с однобайтовым кодом, а при W=1 — с двухбайтовым. Занимающий 6-й и 7-й разряды параметр mod определяет местоположение операнда и уточняет структуру команды, включает она дополнительные байты смещения или нет. Если $mod = 11_2$, то операнд находится в регистре процессора, и тогда параметр r/m определяет его код. В противном случае, при $mod \neq 11_2$, операнд находится в оперативной памяти, а параметр r/m уточняет способ косвенной адресации. В таблице 4.2 представлены двоичные коды регистров процессоро i8086.

Таблица 4.2 – Двоичные коды регистров

000_2	001_2	010_2	011_2	100_2	101_2	110_2	111_2
ax	bx	cx	dx	sp	bp	si	sd

Случай mod ≠ 11_2

Если параметр mod ≠ 11_2, то операнд находится в оперативной памяти. Физический (исполнительный) адрес A поля памяти вычисляется по соотношению

$$A=[S]*10_{16}+Ds, \qquad (4.1)$$

где [S] — содержимое сегментного регистра ss или ds; Ds — внутрисегментное смещение, которое задается как сумма не более чем трех компонентов:

$$Ds=[B]+[I]+Dk, \qquad (4.2)$$

где [B] — содержимое базового регистра bx или bp; [I] — содержимое индексного регистра si или di; Dk — смещение, заданное в команде одним или двумя дополнительными байтами. Каждое из слагаемых в выражении для внутрисегментного смещения может отсутствовать, но не все сразу. Заданное таким образом внутрисегментное смещение D принято называть эффективным адресом.

Если внутрисегментное смещение зависит от регистра bp, то для определения физического адреса по умолчанию выбирается сегмент стека, адрес которого находится в регистре ss. В остальных случаях выбирается сегмент данных с адресом, находящимся в регистре ds.

Набор слагаемых, входящих в выражение для эффективного адреса, определяется кодировкой параметров mod и r/m. При этом параметр mod регулирует наличие или отсутствие слагаемом Dk, а параметр r/m отвечает за комбинацию индексных и базовых регистров, содержимое которых включается в выражение.

В таблице 4.3 представлена используемая кодировка параметра mod. А в таблице 4.4 приведены кодировка параметра r/m при mod ≠ 11_2 и соответствующие этим кодам регистры процессора, которые участвуют в формировании адреса.

Таблица 4.3. Кодировка параметра mod

mod	Длина команды	Пояснение
00_2	2 или 3 байта	Операнд в поле памяти, дополнительное смещение в команду не включается и слагаемое Dk в выражении (4.2) отсутствует
01_2	3 или 4 байта	Операнд в поле памяти, смещение в команде Dk занимает один дополнительный байт
10_2	4 или 5 байтов	Операнд в поле памяти, смещение в команде Dk занимает два дополнительных байта
11_2	2 байта	Операнд в регистре процессора

Таблица 4.4 – Кодировка параметра r/m при mod≠11_2

r/m	B	I	S
000_2	bx	si	ds
001_2	bx	di	ds
010_2	bp	si	ss
011_2	bp	di	ss
100_2	–	si	ds
101_2	–	di	ds
110_2	bp	–	ss
111_2	bx	–	ds

Задание к лабораторной работе:

1. Изучить приведенный теоретический материал к лабораторной работе.
2. Разработать программу на ассемблере в EMU86, использующую все режимы адресации.
3. Из объектного файла выписать соответствующие машинные коды и представить разбор по полям: КОП, КОПд, mod, W, r/m и т.д.

4. Выполнить задание по обработке одномерных массивов, используя базовую и индексную адресации, в соотвествии с таблицей 4.5

Таблица 4.5 – Таблица вариантов

Вариант	**Задание**
1	Найти количество чисел, принадлежащих промежутку [a,b], и сумму чисел, стоящих на местах, кратных 3.
2	Найти сумму чисел, меньших заданного D, и количество чисел, стоящих на четных местах и больших заданного C.
3	Найти произведение всех чисел, стоящих на местах, кратных 4, и количество чисел, небольших заданного A.
4	Найти количество чисел, меньших заданного X, и произведение всех отрицательных чисел, стоящих на нечетных местах.
5	Найти количество чисел, непринадлежащих промежутку (X,Y], и сумму отрицательных чисел, стоящих на четных местах.
6	Найти количество неотрицательных чисел и определить сумму чисел, стоящих на местах, кратных 3, и неравных заданному F.
7	Найти среднее арифметическое отрицательных чисел и определить количество чисел, по величине больших A и стоящих на четных местах.
8	Найти среднее арифметическое положительных чисел, стоящих на нечетных местах, и количество чисел, меньших заданного B.
9	Найти среднее арифметическое чисел, принадлежащих промежутку [A,B), и количество положительных чисел, стоящих на местах, кратных 4.
10	Найти среднее арифметической чисел, неравных заданному C, и произведение неположительных чисел, стоящих на четных местах.
11	Найти среднее арифметическое чисел, больших заданного D и стоящих на нечетных местах, и определить количество чисел, небольших заданного F.
12	Найти среднее арифметическое чисел, непопадающих в промежуток [A,B], и количество положительных чисел, стоящих на местах, кратных 3.
13	Найти среднее арифметическое ненулевых чисел и количество чисел, по величине небольших A и стоящих на четных местах.
14	Вычислить произведение чисел, принадлежащих промежутку (A,B], и количество отрицательных чисел, стоящих на местах,

Вариант	Задание
	кратных 3.
15	Найти среднее арифметическое положительных чисел, стоящих на нечетных местах, и произведение чисел, меньших заданного С.

Требования по содержанию отчета

Отчет должен включать следующие пункты:
1. Титульный лист.
2. Цель работы.
3. Задания на лабораторную работу.
4. Ассемблерный код программ задания 2.
5. Машинные коды с пояснениями.
6. Листинг и результат выполнения программы задания 4;
7. Выводы.

Контрольные вопросы для зашиты:

1. Какие составные части представлены в машиной команде?
2. Какие виды адресации используются в машинных командах?
3. Какая разница между регистровым форматом машинной команды и форматом в общей модификации?
4. Какие двоичные коды используются для обозначения регистров процессора?
5. Что определяет значение поля mod?
6. Каков синтаксис команд ассемблера?
7. Что такое исполнительный адрес и как он используется при определении физического адреса операнда?
8. В чем заключается различие прямых и косвенных режимов адресации?
9. Как различить в командах ассемблера прямые и косвенные режимы адресации?
10. Какие регистры используются в формировании исполнительного адреса при базовой, индексной и непосредственной адресации?

Лабораторная работа № 5
Операции ввода-вывода

Цель работы: изучить способы организации ввода-вывода, используя прерывания BIOS и DOS, получить навыки написания фрагментов программ ввода-вывода.

Краткие теоретические сведения:

Операции ввода-вывода по прерыванию INT 21H

AH = 01: Ввод с клавиатуры с эхоотображением. Данная функция возвращает значение в регистре AL. Если содержимое AL не равно нулю, то оно представляет собой стандартный ASCII-символ, например, букву или цифру. Нулевое значение в регистре AL свидетельствует о том, что на клавиатуре была нажата специальная клавиша, например, Home, F1 или PgUp. Для определения скан-кода клавиш необходимо повторить вызов функции. Данная функция реагирует на запрос Ctrl/break.

AH = 02: Вывод символа. Для вывода символа на экран в текущую позицию курсора необходимо поместить код данного символа в регистр DL. Коды табуляции, возврата каретки и перевода строки действуют обычным образом.

AH = 06: Ввод-вывод данных. Может использоваться как для ввода, так и для вывода. Для вывода занесите в DL выводимый символ (но не FFH!) и прерывание 21H. Для ввода в DL занесите FFH, выполните прерывание 21H. Программа при этом не останавливается, продолжает выполняться. При нажатии клавишы символ вводится в AL.

AH = 07: Прямой ввод с клавиатуры без эхоотображения. Данная функция работает аналогично функции 01 с двумя отличиями: введенный символ не отображается на экране, т.е. нет эха, и отсутствует реакция на запрос Ctrl/Break.

AH = 08: Ввод с клавиатуры без эхоотображения. Данная функция действует аналогично функции 01 с одним отличием: введенный символ не отображается на экране, т.е. нет эха.

AH = 09: Вывод строки символов. Выводимая строка должна заканчиваться знаком доллара $. Адрес начала строки должен быть помещен в DX. Знак доллара не выводится.

Пример:

```
String DB “Введите символ$”      ;Строка для вывода
Mov dx, offset string            ;Смещение строки в регистр DX
Mov  ah, 9       ;               ;Выполняем функцию вывода строки
Int 21h                          ;Прерываемся для вывода строки
```

AH=0AH: Ввод данных в буфер. Определяется максимальная длина вводимого текста. Это необходимо для предупреждения пользователя звуковым сигналом, если набран слишком длинный текст; символы, превышающие максимальную длину, не принимаются. Во второй байт буфера команда возвращает действительную длину введенного текста в байтах. Адрес буфера помещается в регистр DX.

Ниже приведен пример, в котором определен список параметров для области ввода. Первый байт содержит максимальную длину вводимых данных. Так как это однобайтовое поле, то возможное максимальное значение его - шестнадцатеричное FF или 255. Второй байт необходим для занесения в него действительного числа введенных символов. Третьим байтом начинается поле, которое будет содержать введенные символы.

```
MAXLEN DB    20     ; максимальная длина
ACTLEN DB    ?      ; реальная длина
NAMEFLD DB 20 DUP (' ') ; введенные символы
```

Для запроса на ввод необходимо поместить в регистр AH номер функции – 10 (0AH), загрузить адрес списка параметров (MAXLEN в нашем примере) в регистр DX и выполнить INT 21H:

```
MOV AH, 0AH          ; запрос функции ввода
LEA DX, MAXLEN       ; загрузить адрес буфера в DX
INT 21H              ; вызвать системную программу DOS
```

Команда INT ожидает, пока пользователь не введет с клавиатуры текст, проверяя при этом, чтобы число введенных символов не превышало максимального значения, указанного в списке параметров

(20 в нашем примере). Для указания конца ввода пользователь нажимает клавишу Return. Код этой клавиши (0D) также заносится в поле ввода (NAMEFLD в нашем примере). Если, например, пользователь ввел имя BROWN (Return), то список параметров будет содержать информацию:

Десятичные и символьные: 20 5 B R O W N #
Шестнадцатеричные: 14 05 42 52 4F 57 4E OD 20 ...

Во второй байт списка параметров (ACTLEN в нашем примере) команда заносит длину введенного имени – 05. Код Return находится по адресу NAMEFLD + 5. Символ # использован здесь для индикации конца данных, так как 0DH не имеет отображаемого символа. Поскольку максимальная длина в 20 символов включает 0DH, то действительная длина вводимого текста может быть только 19 символов.

AH = 0BH: Проверка состояния клавиатуры. Данная функция возвращает шестнадцатеричное значение FF в регистре AL, если ввод с клавиатуры возможен, в противном случае – 00. это средство связано с функциями 01, 07, 08, которые не ожидают ввода с клавиатуры.

Ввод-вывод по прерыванию INT 10H

AH=0: Установка режима дисплея. В AL указывается номер режима:

AL=0: Текстовый черно-белый 40x25
AL=1: Текстовый цветной 40x25
AL=2: Текстовый черно-белый 80x25
AL=3: Текстовый цветной 80x25
AL=4: Графический цветной 320x200

Пример: установить цветной графический режим:

```
MOV AH,0
MOV AL,4
INT 10H
```

AH=6: Очистка окна (скроллинг вверх)

```
MOV AH,6   ;задаем процедуру скроллинга вверх
```

```
MOV AL,0   ;очищаем все окно
MOV BH,7   ;байт атрибутов для заполнения
MOV CH,3   ;строка верхнего левого угла
MOV CL,4   ;столбец верхнего левого угла
MOV DH,13  ;строка нижнего правого угла
MOV DL,15  ;столбец нижнего правого угла
INT 10H
```

AH=2: Установка курсора в заданную позицию.

```
MOV AH,2    ;устанавливаем номер функции
MOV BH,0    ;номер активной страницы
MOV DH,13   ;строка установки курсора
MOV DL,20   ;столбец установки курсора
INT 10H     ;позиционируем курсор
```

AH=9: Вывод символа с атрибутами на экран

Ввод с клавиатуры по прерыванию BIOS INT 16H

Команда BIOS INT 16H выполняет специальную операцию, которая в соответствии с кодом в регистре AH обеспечивает следующие три функции ввода с клавиатуры.

AH = 00: Чтение символа. Данная функция помещает в регистр AL очередной ASCII-символ, введенный с клавиатуры, и устанавливает скан-код в регистре AH. Если на клавиатуре нажата одна из специальных клавиш, например, Home или F1, то в регистр AL заносится 00. Автоматическое эхо символа на экран не происходит.

AH = 01: Определение наличия введенного символа. Данная функция сбрасывает флаг нуля (ZF=0), если имеется символ для чтения с клавиатуры; очередной символ и скан-код будут помещены в регистры AL и AH соответственно и данный элемент останется в буфере.

AH = 02: Определение текущего состояния клавиатуры. Данная функция возвращает в регистре AL состояние клавиатуры из адреса памяти 417H:

Значения битов результата:

7 Состояние вставки активно (Ins)

6 Состояние фиксации верхнего регистра (Caps Lock) включено

5 Состояние фиксации цифровой клавиатуры (Num Lock) включено

4 Состояние фиксации прокрутки (Scroll Lock) включено
3 Нажата комбинация клавиш Alt/Shft
2 Нажата комбинация клавиш Ctrl/Shft
1 Нажата левая клавиша Shift
0 Нажата правая клавиша Shift

Задание к лабораторной работе:

1. Необходимо написать программу на языке ассемблере с исрользованием операций ввода-вывода, которая выводит название страны и флаг этой страны в соответствии с заданием (таблица 5.1).
2. В каждом задании предлагается ввести данные для вывода информации (пользователь вводит координаты левого верхнего угла и размеры флага, а также название страны).
3. Эскизы флагов находятся в архиве на учебном портале.

Таблица 5.1

Вариант	Страна, флаг	Вариант	Страна, флаг
1.	Армения	16.	Ирландия
2.	Австрия	17.	Италия
3.	Бельгия	18.	Литва
4.	Боливия	19.	Люксембург
5.	Болгария	20.	Мали
6.	Чад	21.	Нидерланды
7.	Колумбия	22.	Норвегия
8.	Чехия	23.	Перу
9.	Финляндия	24.	Румыния
10.	Франция	25.	Россия
11.	Габон	26.	Швеция
12.	Грузия	27.	Швейцария
13.	Германия	28.	Танзания
14.	Гвинея	29.	Таиланд
15.	Венгрия	30.	Украина

Требования по содержанию отчета:

Отчет должен включать следующие пункты:
1. Титульный лист.
2. Цель работы.
3. Задания на лабораторную работу.

4. Листинг программы.
5. Результаты выполнения программы.
6. Выводы.

Контрольные вопросы для защиты:

1. Что такое прерывание?
2. Что делает команда INT 21H?
3. Что делает команда INT 10H?
4. Что делает команда 16H?
5. Как задается функция для выполнения прерывания?
6. Куда вводятся символы при нажатии клавиши Return?
7. Что надо выполнить для вывода символа?
8. Что надо выполнить для ввода символа?
9. Как вводится строка символов?
10. Как установить режим работы дисплея?

Лабораторная работа № 6
Математический сопроцессор

Цель работы: ознакомиться с командами математического сопроцессора, получить навыки программирования математического сопроцессора на языке ассемблер.

Краткие теоретические сведения:

Важной частью микропроцессоров является наличие устройства для обработки числовых данных в формате с плавающей точкой. Сопроцессор (FPU) дополняет основной процессор следующими новыми возможностями по обработке вещественного формата данных:

- полная поддержка стандартов IEEE -754 и - 854 на арифметику с плавающей точкой. Эти стандарты описывают как форматы данных, с которыми должен работать сопроцессор, так и набор реализуемых им функций;
- поддержка численных алгоритмов для вычисления значений тригонометрических функций, логарифмов и т.п. без необходимости самостоятельной разработки соответствующих функций;
- обработка десятичных чисел с точностью до 18 разрядов, что позволяет сопроцессору выполнять арифметические операции без округления над целыми десятичными числами со значениями до 1018;
- обработка вещественных чисел из диапазона 3.37x10-4932...1.18x10+4932.

Набор регистров

С точки зрения программиста, сопроцессор представляет собой совокупность регистров, изображенную на рисунке 6.1, каждый из которых имеет свое функциональное назначение, набор типов данных и систему команд.

В программной модели сопроцессора можно выделить три группы регистров:

- восемь регистров R0 – R7, составляющих основу программной модели сопроцессора – стек сопроцессора;

– три служебных регистра: регистр состояния сопроцессора SWR, управляющий регистр сопроцессора CWR, регистр слова тегов TWR;
– два регистра указателей: данных – DPR и команд – IPR.

Рисунок 6.1 – Программная модель сопроцессора

Размерность каждого из регистров *R0* – *R7* 80 битов. Регистровый стек сопроцессора организован по принципу кольца. Это означает, что все регистры стека с функциональной точки зрения абсолютно одинаковы и равноправны. Как и в любом стеке, в стеке сопроцессора есть вершина, которая является «плавающей». Контроль текущей вершины осуществляется аппаратно с помощью трехбитового поля *TOP* регистра *SWR* (см. рисунок 6.1). В этом поле фиксируется физический номер (0 ... 7) регистра стека, который в данный момент является текущей вершиной стека.

Команды сопроцессора не оперируют физическими номерами регистров стека *R0 ... R7*. Вместо этого они используют логические номера этих регистров *ST(0) ... ST(7)* . С помощью логических номеров реализуется относительная адресация регистров стека сопроцессора.

Система команд сопроцессора

Команды передачи данных

Группа команд передачи данных предназначена для организации обмена между регистрами стека, вершиной стека сопроцессора и ячейками оперативной памяти. С помощью этих команд осуществляются все перемещения значений операндов в сопроцессор и из него.

В сопроцессоре имеются следующие команды передачи данных:

fld/fild источник – загрузка вещественного/целого числа из памяти в вершину стека сопроцессора;

fst/fist приемник – сохранение вещественного/целого числа из вершины стека сопроцессора в память. Сохранение числа в памяти не сопровождается выталкиванием его из стека, то есть текущая вершина стека сопроцессора не изменяется (поле top не изменяется);

fstp/fistp приемник – сохранение вещественного/целого числа из вершины стека со процессора в память. В отличие от предыдущей команды происходит выталкивание вещественного числа из стека после его сохранения в памяти. Команда изменяет поле *TOP*, увеличивая его на единицу. Вследствие этого вершиной стека становится следующий, больший по своему физическому номеру, регистр стека сопроцессора;

fxch st(i) – обмен значений между текущей вершиной стека и регистром стека сопроцессора *ST(i)*.

Пример:

```
double x = 2.7, y = 1.5;
int a = 67, b;

__asm {
        fld  x ;st(0) = x
        fld  y ;st(0) = y, st(1) = x
        fstp  x ;x = st(0), st(0) = x
        fst  y ;y = st(0)
        fistp b ;c = st(0) (с округлением)
        fild  a ;st(0) = a
}
```

Команды загрузки констант

В математических вычислениях достаточно часто встречаются предопределенные константы. Сопроцессор хранит значения некоторых из них. Для каждой предопределенной константы существует своя специальная команда:

fldz – загрузка нуля в вершину стека сопроцессора;

fld1 – загрузка единицы в вершину стека сопроцессора;

fldpi – загрузка числа π в вершину стека сопроцессора;

fldl2t – загрузка двоичного логарифма десяти ($log_2 10$) в вершину стека сопроцессора;

fldl2e – загрузка двоичного логарифма e ($log_2 e$) в вершину стека сопроцессора;

fldlg2 – загрузка десятичного логарифма двух (*lg2*) в вершину стека сопроцессора;

fldln2 – загрузка натурального логарифма двух (*ln2*) в вершину стека сопроцессора.

Команды сравнения данных

Команды данной группы выполняют сравнение значений числа, находящегося в вершине стека и операнда, указанного в команде:

fcom/ficom **[операнд]** – команда без операндов сравнивает два значения: одно находится в регистре *ST(0)* , другое – в регистре *ST(1)* . Если указан операнд, то сравнивается значение в регистре *ST(0)* стека сопроцессора со значением в памяти. По результатам сравнения устанавливаются биты *C3, C2, C0* регистра *SWR*;

fcomp/ficomp **[операнд]** – команда сравнивает значение в вершине стека сопроцессора *ST(0)* со значением операнда, который находится в регистре или в памяти. После сравнения и установки бит *C3, C2, C0* команда выталкивает значение из *ST(0)*. Длина целого операнда 16 или 32 бита, то есть целое слово и короткое целое;

ftst – команда не имеет операндов и сравнивает значение в *ST(0)* со значением 0.

В результате работы приведенных команд сравнения в регистре состояния *SWR* устанавливаются биты *C3, C2, C0* как указано в таблице 6.1.

Таблица 6.1 – Результаты выполнения сравнения

Результат операции	**C3**	**C2**	**C0**
операнды несравнимы	1	1	1
операнд_2 > операнд_1 (ST(1) > ST(0))	0	0	1
операнд_2 < операнд_1 (ST(1) < ST(0))	0	0	0
операнд_2 = операнд_1 (ST(1) = ST(0))	1	0	0

Для программирования реакции на результаты сравнения необходимо анализировать состояние этих битов. Сопроцессор не имеет для этого специальных команд, за исключением команды *fstsw*, которая позволяет записать содержимое регистра *SWR* в регистр *AX*. Затем командой *sahf* содержимое регистра *AH*, в котором находятся биты *C3, C2 и C0*, записывается в младший байт регистра *EFLAGS/FLAGS*. Местоположение бит *C3, C2 и C0* соответствует местоположению флагов *ZF, PF* и *CF*. Таким образом, после применения команды *sahf* становится возможным реагировать на результаты сравнения значений в сопроцессоре командами условного перехода основного процессора.

Например, приведенная ниже функция возвратит 0 если a и b несравнимы, 1 если a < b, 2 если a > b и 3 если a = b.

```
int compare_real(double a, double b)
{
int r = 0;
__asm {
        fld  a
        fld  b
        fcom
        fstsw ax
        sahf   ;cf = c3, pf = c2, zf = c0
        jp  not_cmp ; //если числа несравнимы
        jc  if_less  ; //если a < b
        jz  if_equal  ; //если a = b
        jmp  if_great  ; //если a > b
        not_cmp:
        xor  eax, eax  ; //вернуть 0
        jmp  end_cmp
        if_less:
```

```
        mov  eax, 1; //вернуть 1
        jmp  end_cmp
        if_great:
        mov  eax, 2 ; //вернуть 2
        jmp  end_cmp
        if_equal:
        mov  eax, 3  ; //вернуть 3
        end_cmp:
        mov  r, eax
}
return r;
}
```

Арифметические команды

Команды сопроцессора, входящие в данную группу, реализуют четыре основные арифметические операции – сложение, вычитание, умножение и деление. Имеется также несколько дополнительных команд, предназначенных для повышения эффективности использования основных арифметических команд.

Целочисленные арифметические команды:

fiadd источник – команда складывает значения *ST(0)* и целочисленного источника, в качестве которого выступает 16- или 32-разрядный операнд в памяти. Результат сложения запоминается в регистре стека сопроцессора *ST(0)*;

fisub источник – команда вычитает значение целочисленного источника из *ST(0)*. Результат вычитания запоминается в регистре стека сопроцессора *ST(0)*. В качестве источника выступает 16- или 32-разрядный целочисленный операнд в памяти;

fimul источник – команда умножает значение целочисленного источника на содержимое *ST(0)* . Результат умножения запоминается в регистре стека сопроцессора *ST(0)*. В качестве источника выступает 16- или 32-разрядный целочисленный операнд в памяти;

fidiv источник – команда делит содержимое *ST(0)* на значение целочисленного источника. Результат деления запоминается в регистре стека сопроцессора *ST(0)*. В качестве источника выступает 16- или 32-разрядный целочисленный операнд в памяти.

Вещественные арифметические команды:

fadd – команда складывает значения из регистров *ST(0)* и *ST(1)*. Результат сложения запоминается в регистре стека сопроцессора *ST(0)*;

fadd слагаемое – команда складывает значения *ST(0)* и слагаемого, представляющего собой адрес ячейки памяти. Результат сложения запоминается в регистре стека сопроцессора *ST(0)*;

fadd st(i), st – команда складывает значение в регистре стека сопроцессора *ST(i)* со значением в вершине стека *ST(0)*. Результат сложения запоминается в регистре *ST(i)* ;

faddp st(i), st – команда производит сложение вещественных операндов аналогично команде *fadd st(i), st*. Последним действием команды является выталкивание значения из вершины стека сопроцессора *ST(0)* . Результат сложения остается в регистре *ST(i-1);*

fsub – команда вычитает из значения в регистре *ST(1)* значение регистра *ST(0)*. Результат вычитания запоминается в регистре стека сопроцессора *ST(0);*

fsub вычитаемое – команда вычитает значение вычитаемого из значения в *ST(0)*. Вычитаемое представляет собой адрес ячейки памяти, содержащей допустимое вещественно е число. Результат сложения запоминается в регистре стека сопроцессора *ST(0)*;

fsub st(i), st – команда вычитает значение в вершине стека ST(0) из значения в регистре стека сопроцессора *ST(i).* Результат вычитания запоминается в регистре стека сопроцессора *ST(i)*;

fsubp st(i), st – команда вычитает вещественные операнды аналогично команде *fsubp st(i), st*. Последним действием команды является выталкивание значения из вершины стека сопроцессора *ST(0)*. Результат вычитания остается в регистре *ST(i-1)*;

fmul – команда (без операндов) умножает значение в *ST(1)* на содержимое регистра в *ST(0)*. Результат умножения запоминается в регистре стека сопроцессора *ST(0);*

fmul **множитель** – команда умножает множитель на содержимое регистра стека *ST(0)*. Результат умножения запоминается в регистре стека сопроцессора *ST(0)*;

fmul st(i), st – команда умножает значение в произвольном регистре *ST(i)* на содержимое регистра стека *ST(0)*. Результат умножения запоминается в регистре стека сопроцессора *ST(i)*;

fmulp st(i), st – команда производит умножение подобно команде *fmul st(i), st*. Последним действием команды является

выталкивание значения из вершины стека сопроцессора *ST(0)* . Результат умножения остается в регистре *ST(i-1)*;

fdiv – команда (без операндов) делит содержимое регистра *ST(1)* на значение регистра сопроцессора *ST(0)*. Результат деления запоминается в регистре стека сопроцессора *ST(0)*;

fdiv **делитель** – команда делит содержимое регистра *ST(0)* на делитель. Результат деления запоминается в регистре стека сопроцессора *ST(0);*

fdiv st(i), st – команда производит деление содержимого регистра *ST(i)* на значение регистра сопроцессора *ST(0)* . Результат деления запоминается в регистре стека сопроцессора *ST(i)*;

fdivp st(i), st – команда производит деление аналогично команде *fdiv st(i), st.* Последним действием команды является выталкивание значения из вершины стека сопроцессора *ST(0)*. Результат деления остается в регистре *ST(i-1).*

Дополнительные арифметические команды:

fsqrt – вычисление квадратного корня из значения, находящегося в вершине стека сопроцессора *ST(0)* . Команда не имеет операндов. Результат вычисления помещается в регистр *ST(0)* . Следует отметить, что данная команда обладает определенными достоинствами. Во-первых, результат извлечения достаточно точен, во-вторых, скорость исполнения чуть больше скорости выполнения команды деления вещественных чисел *fdiv*;

fabs – вычисление модуля значения, находящегося в вершине стека сопроцессора – регистре *ST(0)*. Команда не имеет операндов. Результат вычисления помещается в регистр *ST(0)*;

fchs – изменение знака значения, находящегося в вершине стека сопроцессора – регистре *ST(0)*. Команда не имеет операндов. Результат вычисления помещается обратно в регистр *ST(0)* . Отличие команды *fchs* от команды *fabs* в том, что команда *fchs* только инвертирует знаковый разряд значения в регистре *ST(0)*, не меняя значения остальных бит. Команда вычисления модуля *fabs* при наличии отрицательного значения в регистре *ST(0)*, наряду с инвертированием знакового ряда, выполняет изменение остальных бит значения таким образом, чтобы в *ST(0)* получилось соответствующее положительное число;

fxtract – команда выделения порядка и мантиссы значения, находящегося в вершине стека сопроцессора – регистре *ST(0).*

Команда не имеет операндов. Результат выделения помещается в два регистра стека – мантисса в *ST(0)*, а порядок в *ST(1)*. При этом мантисса представляется вещественным числом с тем же знаком, что и у исходного числа, и порядком равным нулю. Порядок, помещенный в *ST(1)*, представляется как истинный порядок, то есть без константы смещения, в виде вещественного числа со знаком и соответствует величине *p* формулы $A = (\pm M) \cdot N^{\pm(p)}$;

fscale – команда умножает или делит значение, находящееся в вершине стека сопроцессора *ST(0)*, на степень двух. Значение показателя степени находится в регистре стека *ST(1)* и представляет собой целое число со знаком. Если показатель степени не является целым числом, то он округляется до целого в меньшую сторону. Команда не имеет операндов. Команда *fscale* не очищает регистр *ST(1)*. Результат помещается в регистр *ST(0)*;

frndint – команда округления до целого значения – округляет значение, находящееся в вершине стека сопроцессора *ST(0)* . Команда не имеет операндов.

Команды трансцендентных функций

Сопроцессор имеет ряд команд, предназначенных для вычисления значений тригонометрических функций, таких, как синус, косинус, тангенс, арктангенс, а так же значений логарифмических и показательных функций.

Наличие этих команд значительно облегчает разработку вычислительных алгоритмов. В системе команд сопроцессора имеются следующие трансцендентные команды:

fcos – команда вычисляет косинус угла в радианах, находящегося в вершине стека сопроцессора *ST(0)*. Команда не имеет операндов. Результат возвращается в регистр *ST(0)*;

fsin – команда вычисляет синус угла в радианах, находящегося в вершине стека сопроцессора *ST(0)*. Команда не имеет операндов. Результат возвращается в регистр *ST(0)*;

fsincos – команда вычисляет синус и косинус угла в радианах, находящегося в вершине стека сопроцессора *ST(0)*. Команда не имеет операндов. Результат возвращается в регистры *ST(0)* и *ST(1)* . При этом косинус помещается в *ST(0)* , а синус в *ST(1)* ;

fptan – команда вычисляет частичный тангенс угла в радианах, (в диапазоне –263 … 263), находящегося в вершине стека сопроцессора *ST(0)*. Команда не имеет операндов. Результат

возвращается в регистры *ST(0)* – значение единицы, и *ST(1)* – тангенс угла;

fpatan – команда вычисляет частичный арктангенс частного *x/y* (*x* находится в регистре *ST(0)* , *y* – в регистре *ST(1)*). Команда не имеет операндов. Результат возвращается в регистре *ST(0)* ;

fprem – команда получения частичного остатка от деления. Исходные значения делимого и делителя размещаются в стеке – делимое в *ST(0)* , делитель в *ST(1)*. Делитель рассматривается как некоторый модуль, поэтому в результате работы команды получается остаток от деления по модулю. Но произойти это может не сразу, так как этот результат в общем случае достигается за несколько производимых подряд обращений к команде *fprem*. Это происходит, если значения операндов сильно различаются. Физическая работа команды заключается в реализации хорошо известного действия деления в столбик. При этом каждое промежуточное деление осуществляется отдельной командой *fprem*. Цикл, центральное место в котором занимает команда *fprem*, завершается, когда очередная полученная разность в *ST(0)* становится меньше значения модуля в *ST(1)*. Судить об этом можно по состоянию флага *C2* в регистре состояния *SWR*: если *C2=0*, то работа команды *fprem* полностью завершена, так как разность *ST(0)* меньше значения модуля в *ST(1)*; если *C2=1*, то необходимо продолжить выполнение команды *fprem*, так как разность *ST(0)* больше значения модуля в *ST(1);*

f2xm1 – команда вычисления значения функции $y = 2^x - 1$. Исходное значение *x* размещается в вершине стека сопроцессора в регистре *ST(0)* и должно лежать в диапазоне $-1 \le x \le 1$. Результат *y* замещает *x* в регистре *ST(0)*. Эта команда может быть использована для вычисления различных показательных функций;

fyl2x – команда вычисления значения функции $z = y \cdot log_2 x$. Исходное значение *x* размещается в вершине стека сопроцессора, а исходное значение *y* – в регистре *ST(1)*. Значение *x* должно лежать в диапазоне $0 \le x \le +\infty$, а значение *y* – в диапазоне $-\infty \le y \le +\infty$. Перед тем, как осуществить запись результата *z* в вершину стека, команда *fyl2x* выталкивает значения *x* и *y* из стека и только после этого производит запись *z* в регистр *ST(0);*

fyl2xp1 – команда вычисления значения функции $z = y \cdot log_2(x+1)$. Исходное значение *x* размещается в вершине стека сопроцессора – регистре *ST(0)*, а исходное значение *y* – в регистре *ST(1)*. Значение *x* должно лежать в диапазоне $0 \le x \le 1-1/\sqrt{2}$,

а значение y – в диапазоне - $-\infty \leq y \leq +\infty$. Перед тем, как осуществить запись результата *z* в вершину стека, команда *fyl2xp1* выталкивает значения *x* и *y* из стека и только после этого производит запись *z* в регистр *ST(0)*.

Пример: необходимо вычислить значение следующего выражения $z = x \log4(y) ctg(y+x)$ при $x=1.32$ и $y=4.19$.

```
double x = 1.32, y = 4.19, z, ol = 4.0;
__asm {
        finit;
        fld1   ;
        fld  ol ;
        fyl2x  ; //st(0) = 1*log2(4) = t1
        fld  x ;
        fld  y ;
        fyl2x  ; //st(0) = x*log2(y) = t2, st(1) = t1
        fdiv   ; //st(0) = t2/t1 = t3
        fld  x ;
        fld  y ;
        fadd   ; //st(0) = x+y = t4, st(1) = t3
        fptan  ; //st(0) = 1, st(1) = tg(t4), st(2) = t3
        fdiv   ; //st(0) = 1/tg(t4) = t5, st(1) = t3
        fmul   ; //st(0) = t5*t3
        fstp  z ; //z = x*log4(y)*ctg(x+y)
}
printf("z = %lf \n", z);
```

Команды управления сопроцессором

Последняя группа команд предназначена для общего управления работой сопроцессора:

wait/fwait – команда ожидания, предназначена для синхронизации работы процессора и сопроцессора;

finit – команда инициализации сопроцессора. Данную команду используют перед первой командой сопроцессора в программе и в других случаях, когда необходимо привести сопроцессор в начальное состояние;

fstsw **назначение** – команда сохранения содержимого регистра состояния *SWR* в ячейке памяти размером в 2 байта или регистре *AX*;

***fstcw* приемник** – команда сохраняет содержимое регистра управления сопроцессора в 16-битную ячейку памяти, адресуемую операндом приемник. Команда позволяет получить доступ к части информации рабочей среды сопроцессора, а именно, к содержимому регистра *CWR*. В зависимости от особенностей конкретного алгоритма интерес представляют все поля регистра *CWR* – маски исключений, управление точностью и округлением. После их анализа можно установить новые значения полей и с помощью команды fldcw записать обновленное слово обратно в регистр CWR;

***fldcw* источник** – команда выполняет операцию загрузки значения регистра *CWR* из 16-битной ячейки памяти, адресуемой операндом источник. Это позволяет изменить определенные режимы работы сопроцессора, например, размаскировать нужные исключения;

ffree st(i) – команда освобождения регистра стека *ST(i)*. Команда записывает в поле регистра тегов, соответствующего регистру *ST(i)*, значение 11b, что соответствует пустому регистру.

Пример преобразования вещественного числа в целое путем отбрасывания дробной части:

```
double d = 123.456;
int x;
short cw;
__asm {
        finit
        fstcw cw
        or  cw, 0000110000000000b
        fldcw cw
        fld  d
        fist  x
        finit
}
```

Задание к лабораторной работе:

Необходимо разработать консольное приложение, которое вычисляет заданную (в соответствии с вариантом) функцию двумя способами:

– с использованием команд сопроцессора;
– с использованием стандартной библиотеки math.h.

После вычислений должно быть выведено время выполнения для каждого случая.

Функция *F(x)* вычисляется для значений x в интервале от a до b с шагом d, где a, b и d вводятся с клавиатуры.

В программе должна быть предусмотрена возможность многократного выполнения вычислений без выхода из программы.

Таблица 6.2

Номер вариант	**Функция**	**Номер варианта**	**Функции**
1	F(x) = sin(x) * sqrt(|x|)	16	F(x) = sin(x)/(x^2+10)
2	F(x) = x / |sqrt(|x|) + 1|	17	F(x) = sin(x)*tg(x+5)
3	F(x) = x^3/(x^2+1)	18	F(x) = sqrt(x)/(x^2+1)
4	F(x) = (x + 1) / (x^2 + 1)	19	F(x) = |sin^2(x) * cos(x)|
5	F(x) = (x^2 + x) / |x - 1| , **(x != 1)**	20	F(x) = x^2 – x + sqrt(|x|)
6	F(x) = sin(x^2) + ½	21	F(x) = sin(x^2 + 5) * 4
7	F(x) = |sin(x) – sqrt(x)|	22	F(x) = sin (cos (sqrt (x) + 1)), **x >= 0**
8	F(x) = 2sin(x) + cos(3x)	23	F(x) = x^2 – sin (sqrt (x)), **x >= 0**
9	F(x) = x / (|sin(x)| + 2)	24	F(x) = x^2 + cos (sqrt (x)), **x >= 0**
10	F(x) = x * sqrt(x)/log2(x) , **x > 0**	25	F(x) = (2*x + 1) / 3 + (x^2 + 1)
11	F(x) = x – x^2*cos(x)	26	F(x) = (x*7 + 1) * (x^2 - 2)
12	F(x) = x^4 – x^3 + x^2 – x + 1	27	F(x) = (cos(x) + 1) / (sin(x) + 4)
13	F(x) = sin(x) * cos(x^2)	28	F(x) = tg(x+1)/sin(x) , **x != 0**
14	F(x) = (sqrt (x) - 1)/(x+1), **x >= 0**	29	F(x) = ctg(x-1) * cos(x)
15	F(x) = (($x+1)^2$ - 1)/x, **x != 0**	30	F(x) = 3*log_2(x)*sin(x)

Требования по содержанию отчета:

Отчет должен включать следующие пункты:

1. Титульный лист.
2. Цель работы.
3. Задание.

4. Листинг программного кода с комментариями.
5. Тесты.
6. Результаты выполнения программы.
7. Выводы.

Контрольные вопросы для защиты работ:

1. Какие данные может обрабатывать сопроцессор?
2. Какие команды относятся к командам передачи данных?
3. Какие команды используются для сравнения данных?
4. Какие арифметические команды сопроцессора вы знаете?
5. Назовите регистры сопроцессора и их назначение.
6. Как вычислить тангенс угла с помощью команд сопроцессора?
7. Как вычислить котангенс угла с помощью команд сопроцессора?
8. Как возвести число в степень?
9. Для чего используется математический сопроцессор?
10. Для чего нужна синхронизация сопроцессора с процессором?
11. Какая команда используется для синхронизации работы процессора и сопроцессора?

Лабораторная работа № 7
Использование стековой адресации для вычисления арифметических выражений

Цель работы: изучить возможности стековой адресации для разбора и вычисления арифметического выражения.

Краткие теоретические сведения:

Стековая адресация

В лабораторной работе № 4 мы получили знания о формате машинных команд. Для повышения эффектиивноти выполнения машинных команд, желательно чтобы они были как можно короче. Максимальный эффект достигается при использовании безадресных команд. Безадресные команды возможны при наличии стека. Рассмотрим стековую адресацию более подробно на примере разбора и вычисления арифметического выражения.

Обратная польская запись

В математике существует древняя традиция помещать оператор между операндами *(x + y)*, а не после операндов *(x y +)*. Форма с оператором между операндами называется инфиксной записью. Форма с оператором после операндов называется постфиксной, или обратной польской записью в честь польского логика Я. Лукасевича (1958), который изучал свойства этой записи.

Обратная польская запись имеет ряд преимуществ перед инфиксной записью при выражении алгебраических формул. Во-первых, любая формула может быть выражена без скобок. Во-вторых, она удобна для вычисления формул в машинах со стеками. В-третьих, инфиксные операторы имеют приоритеты, которые произвольны и нежелательны.

Существует несколько алгоритмов для превращения инфиксных формул в обратную польскую запись. Мы рассмотрим алгоритм, предложенный Э. Дейкстра (E. W. Dijkstra). Для этого вводится понятие стекового приоритета операций (таблица 7.1).

Таблица 7.1

Операция	**Приоритет**
(	0
)	1
+|-	2
*|/	3

Просматривается исходная строка символов слева направо, операнды переписываются в выходную строку, а знаки операций заносятся в стек на основе следующих соображений:

- если стек пуст, то операция из входной строки переписывается в стек;
- операция выталкивает из стека все операции с большим или равным приоритетом в выходную строку;
- если очередной символ из исходной строки есть открывающая скобка, то он проталкивается в стек;
- закрывающая круглая скобка выталкивает все операции из стека до ближайшей открывающей скобки, сами скобки в выходную строку не переписываются, а уничтожают друг друга.

В обратной польской записи операторы появляются в том порядке, в котором они будут выполняться. В таблице 7.2 привежены примеры инфиксных формул и их эквивалентов в обратной польской записи.

Таблица 7.3 – Примеры инфиксных выражений и их эквиваленты в обратной польской записи

Инфиксная запись	Обратная польская запись
A + B x C	A B C x +
A x B + C	A B x C +
A x B + C x D	A B x C D x +
(A + B) / (C - D)	A B + C D - /
A x B / C	A B x C /
((A + B) x C + D) / (E + F + G)	A B + C x D + E F + G + /

Вычисление формул в обратной польской записи

Обратная польская запись идеально подходит для вычисления формул на компьютере со стеком. Формула состоит из *n* символов, каждый из которых является либо операндом, либо оператором. Алгоритм для вычисления формулы в обратной польской записи с использованием стека прост. Нужно просто прочитать обратную польскую запись слева направо. Если встречается операнд, его нужно поместить в стек. Если встречается оператор, нужно выполнить заданную им операцию. Таблица 7.3 иллюстрирует вычисление следующего выражения: (8 + 2 x 5) / A + 3 x 2 - 4), некоторой виртуальной машиной. Предполагается, в архитектуре набра команд есть безадресные команды IADD, ISUB, IMUL и IDIV, использующие стек для хранения операндов. Отметим, что преобразовать обратную польскую запись в код нашей виртуальной машины очень легко: нужно просто двигаться по формуле в обратной польской записи, записывая по одной команде для каждого символа. Если символ является константой или переменной, нужно вписывать команду помещения этой константы или переменной в стек, если символ является оператором, нужно вписывать команду для выполнения данной операции.

Таблица 7.3 – Использование стека для вычисления формулы в обратной польской записи

Шаг	**Оставшаяся цепочка**	**Команда**	**Стек**
1	8 2 5 x + 1 3 2 x + 4 - /	PUSH 8	8
2	2 5 x + 1 3 2 x + 4 - /	PUSH 2	8, 2
3	5 x + 1 3 2 x + 4 - /	PUSH 5	8, 2, 5
4	x + 1 3 2 x + 4 - /	IMUL	8, 10
5	+ 1 3 2 x + 4 - /	IADD	18
6	1 3 2 x + 4 - /	PUSH 1	18, 1
7	3 2 x + 4 - /	PUSH 3	18, 1, 3
8	2 x + 4 - /	PUSH 2	18, 1, 3, 2
9	x + 4 - /	IMUL	18, 1. 6
10	+ 4 - /	IADD	18, 7
11	4 - /	PUSH 3	18, 7, 4
12	- /	ISUB	18, 3
13	/	IDIV	6

Задание к лабораторной работе:

1. Разработать консольное приложение, предлагающее пользователю ввести арифметическое выражение.
2. Вычисление арифметического выражения должно вычисляться в математическом сопроцессоре.
3. Протестировать разработанную программу задавая в качестве операндов числовые константы, имена переменных, вызовы стандартных функций.
4. Отлаженную программу оформить в виде библиотеки для возможности вызова в любой программе.

Требования по содержанию отчета:

Отчет должен включать следующие пункты:
1. Титульный лист.
2. Цель работы.
3. Задание.
4. Листинг прогрммы.
5. Результаты тестирования программы.
6. Выводы

Контрольные вопросы для защиты:

1. Что такое инфиксная форма записи?
2. В чем отличие постфиксной формы записи?
3. Приведите алгоритм перевода арифметического выражения в обратную польскую запись.
4. Приведите алгоритм вычисления арифметического выражения, представленного в форме обратной польской записи.
5. В чем преимущество стековой адресации?
6. Каков приоритет операции для выталкивания значения из стека?
7. Каков приоритет операции для занесения значения в стек?.
8. Каковы правила работы со скобками?

Лабораторная работа № 8
Управление выводом графической информации на базе библиотеки GDI

Цель работы: изучить основы управления графической информацией на базе библиотеки GDI

Краткие теоретические сведения:

Графический интерфейс устройства (Graphics device interface (GDI)) — это составная часть Win32 API, обеспечивающая графический вывод на устройства отображения информации, такие как дисплей или принтер. Windows-приложение не имеет непосредственного доступа к аппаратным устройствам. Вместо этого оно обращается к функциям GDI, а GDI транслирует эти обращения к программным драйверам физических устройств, обеспечивая аппаратную независимость приложений. Код библиотеки GDI находится в файле gdi32.dll, то есть библиотека является динамически загружаемой. Драйверы стандартных устройств поставляются как часть Windows, а драйверы специализированных устройств создаются производителями оборудованиями.

Контекст устройства

Взаимодействие приложения с GDI осуществляется при непременном участии еще одного посредника — так называемого контекста устройства.

Контекст устройства (Device Context (DC)) — это внутренняя структура данных, которая определяет набор графических объектов и ассоциированных с ними атрибутов, а также графических режимов, влияющих на вывод.

Ниже приведен список основных графических объектов:

- перо (pen) для рисования линий;
- кисть (brush) для заполнения фона или заливки фигур;
- растровое изображение (bitmap) для отображения в указанной области окна;
- палитра (palette) для определения набора доступных цветов;
- шрифт (font) для вывода текста;
- регион (region) для отсечения области вывода.

Если необходимо рисовать на устройстве графического вывода (экране дисплея или принтере), то сначала нужно получить дескриптор контекста устройства. Возвращая этот дескриптор после вызова соответствующих функций, Windows тем самым предоставляет разработчику право на использование данного устройства. После этого дескриптор контекста устройства передается как параметр в функции GDI, чтобы идентифицировать устройство, на котором должно выполняться рисование.

Контекст устройства содержит много атрибутов, определяющих поведение функций GDI. Благодаря этому списки параметров функций GDI содержат только самую необходимую информацию, например, начальные координаты или размеры графического объекта. Все остальное система извлекает из контекста устройства.

В данной лабораторной работе используется контекст дисплея.

Общие операции с графическими объектами

Как уже упоминалось ранее, GDI содержит набор графических объектов, обеспечивающих выполнение графических операций. К таким объектам относятся перья, кисти, растровые изображения, палитры, шрифты. Использование любого графического объекта предполагает выполнение следующей последовательности операций:

1. Создать графический объект.
2. Выбрать созданный объект в контекст устройства.
3. Вызвать графическую функцию, работающую с объектом.
4. Удалить объект из контекста устройства, вернув предшествующий объект.
5. Уничтожить объект.

Для создания GDI-объектов предназначены соответствующие функции Create..., которые в случае успешного завершения возвращают дескриптор объекта. Выбор объекта в контекст устройства осуществляется при помощи функции SelectObject (палитры выбираются с помощью функции SelectPalette).

Функция SelectObject имеет следующий прототип:

```
HGDIOBJ SelectObject(
        HDC hdc,                // дескриптор контекста устройства
        HGDIOBJ hgdiobj         // дескриптор GDI-объекта
);
```

В результате ее выполнения новый объект hgdiobj заменяет предшествующий объект того же типа в контексте устройства hdc.

Функция возвращает дескриптор предшествующего объекта. Для корректной работы приложение должно запомнить этот дескриптор и после окончания рисования с новым объектом (шаг 3) вернуть в контекст устройства предшествующий объект (шаг 4).

Для уничтожения объекта, ставшего ненужным, вызывается функция DeleteObject.

Линии и кривые

Любая линия рисуется в Windows с использованием графического объекта, называемого пером. Контекст устройства содержит перо по умолчанию − сплошное перо черного цвета толщиной 1 пиксел. Многие графические функции начинают рисование с так называемой текущей позиции пера.

Текущая позиция пера является одним из атрибутов контекста устройства. Она определяется значением типа POINT. По умолчанию текущая позиция пера установлена в точку (0, 0). Если нужно изменить текущую позицию, вызывается функция MoveToEx:

```
BOOL MoveToEx(
      HDC hdc.          // дескриптор контекста устройства
      int X,            // x-координата новой текущей позиции
      int Y,            // y-координата новой текущей позиции
      LPPOINT lpPoint // предыдущая позиция пера
);
```

Если предыдущая позиция пера нас не интересует, можно передать последнему параметру значение NULL. В случае успешного завершения функция возвращает ненулевое значение.

В любой момент можно получить значение текущей позиции пера, вызвав функцию GetCurrentPositionExt(hDC, &pt).

Результат выполнения функции помещается в переменную pt типа POINT.

Для создания прямой линии используется функция LineTo:

```
BOOL LineTo(
      HDC hdc. // дескриптор контекста устройства
```

int xEnd. // x-координата конечной точки
int yEnd // y-координата конечной точки
):

Эта функция рисует отрезок, начиная с точки, в которой находится текущая позиция пера, до точки (xEnd, yEnd), не включая последнюю точку в отрезок.

Координаты конечной точки задаются в логических единицах. Если функция завершается успешно, то она возвращает ненулевое значение, а текущая позиция пера устанавливается в точку (xEnd, yEnd). Последнее свойство функции можно использовать, если требуется нарисовать ряд связанных отрезков. Например, можно определить массив из пяти точек, задающих контур прямоугольника, в котором последняя точка совпадает с первой, и нарисовать прямоугольник.

Последовательность связанных отрезков гораздо проще нарисовать с помощью функции Polyline:

```
BOOL Polyline(HDC hdc, CONST POINT* lppt, int cPoints);
```

Второй параметр здесь – это адрес массива точек, а третий – количество точек.

Функция PolylineTo предназначена для рисования последовательности связанных отрезков:

```
BOOL PolylineTo(HDC hdc, CONST POINT* lppt, DWORD cPoints);
```

Она использует текущую позицию пера для начальной точки и после каждого своего выполнения устанавливает текущую позицию в конец нарисованного отрезка. Если вы захотите применить ее в рассмотренном выше примере, то вызов

Функция PolyPolyline позволяет нарисовать несколько ломаных линий за один вызов. Она имеет следующий прототип:

```
BOOL PolyPolyline(
    HDC hdc,              // дескриптор контекста устройства
    CONST POINT* lppt, // массив точек для нескольких ломаных линий
```

```
    CONST DWORD* lpPolyPoints, // массив с числом точек в
каждой ломаной
    DWORD cCount // количество ломаных линий
);
```

Дуги

Дуги в Windows рисуются как часть эллипса. Размеры и расположение эллипса определяются ограничивающим прямоугольником. Ограничивающий прямоугольник задается координатами левой верхней и правой нижней вершин.

Для рисования дуг предназначены функции Arc, ArcTo и AngleArc. Первые две функции имеют одинаковый набор параметров, поэтому рассмотрим прототип функции Arc:

```
BOOL Arc(HDC hdc, int xLeft. int yTop. int xRight, int yBottom,
int xStart, int yStart, int xEnd, int yEnd);
```

Параметры со второго по пятый задают вершины ограничивающего прямоугольника. Начало и конец дуги определяются начальным и конечным углами, которые задаются косвенно через две дополнительные точки с координатами (xStart, yStart) и (xEnd, yEnd). Начало дуги — это пересечение эллипса с лучом, который начинается в центре эллипса и проходит через точку (xStart, yStart). Конец дуги — это пересечение эллипса с лучом, который начинается в центре эллипса и проходит через точку (xEnd, yEnd). Все координаты задаются в логических единицах. Направление дуги определяется соответствующим атрибутом в контексте устройства, значение которого можно получить вызовом функции GetArcDirection или установить вызовом функции SetArcDirection.

Функция ArcTo отличается от функции Arc тем, что она проводит линию из текущей позиции пера в заданную начальную точку дуги, и только после этого рисует дугу. После завершения рисования функция перемещает текущую позицию пера в конечную точку дуги.

Функция AngleArc существенно отличается от двух предыдущих функций способом определения рисуемой дуги, что выражено и в ее прототипе:

BOOL AngleArc(HDC hdc, int X, int Y, DWORD radius, FLOAT startAngle, FLOAT sweepAngle):

Параметры X и Y задают центр круга, а параметр radius — его радиус. Дуга, рисуемая этой функцией, является частью круга. Чтобы нарисовать часть эллипса, приложение должно определить соответствующее преобразование или отображение.

Параметр startAngle определяет начальный угол дуги в градусах, а параметр sweepAngle – длину дуги в градусах. При положительном значении sweepAngle дуга рисуется против часовой стрелки, при отрицательном значении – по часовой стрелке.

Кривые Безье

В дополнение к рисованию кривых, являющихся частью эллипса, Windows позволяет рисовать нерегулярные кривые, называемые кривыми Безье. Кривая Безье (сплайн Безье) – это кубическая кривая, положение и кривизна которой задаются четырьмя определяющими точками p1, p2, p3 и p4. Точка p1 является стартовой точкой, точка p4 – конечной точкой. Точки p2, p3 называются контрольными точками – именно они определяют форму кривой, играя роль «магнитов», оттягивающих линию от прямой, соединяющей p1 и p4.

GDI содержит две функции, позволяющие рисовать набор связанных кривых Безье за один вызов:

BOOL PolyBezier(HDC hdc, CONST POINT* lppt. DWORD cPoints);

BOOL PolyBezierTo(HDC hdc, CONST POINT* lppt, DWORD cCount);

Для рисования n кривых функция PolyBezier получает адрес массива lppt, содержащего 3n + 1 точек, при этом параметр cPoints должен быть равен 3n + 1. Первые четыре точки, lppt[0], lppt[l], lppt[2], lppt[3], задают первую кривую. Точка lppt[3] вместе со следующими тремя точками определяет вторую кривую и т. д.

В отличие от PolyBezier, функция PolyBezierTo получает 3n точек в массиве lppt, при этом параметр cCount должен быть равен 3n. Функция рисует первую кривую, начиная с текущей позиции пера, до

позиции, заданной третьей точкой, используя первые две точки в качестве контрольных. Каждая последующая кривая рисуется так же, как и функцией PolyBezier. По окончании рисования текущая позиция пера переводится в последнюю точку из массива lppt.

Перья

Любая функция рисования линий и кривых, а также контуров замкнутых фигур использует перо (pen), выбранное в контексте устройства в данный момент. Если вы не выбирали никакого пера, то используется перо по умолчанию BLACK_PEN. Оно рисует сплошные черные линии толщиной 1 пиксел независимо от режима отображения.

Для задания собственых настрое пера Win GDI позволяет создавать объекты логических перьев. Логическое перо представляет собой описание требований к перу со стороны приложения. Эти требования могут в каких-то деталях и не соответствовать тому, как будут выводиться линии на поверхности физического устройства. Драйвер графического устройства может поддерживать собственные

Для дескрипторов логических перьев зарезервирован тип HPEN (handle to a pen). Значение дескриптора получают вызовом соответствующей функции. Вид вызываемой функции зависит от типа пера.

Объект логического пера, как и другие объекты GDI, поглощает ресурсы операционной системы. Следовательно, когда необходимость в нем отпадает, рекомендуется исключить его из контекста устройства и удалить функцией DeleteObject.

Вывод текста

Win GDI обеспечивает полный набор функций для форматирования и рисования текста в клиентской области окна или на бумажной странице принтера.

Простейшая функция вывода текстовой строки TextOut имеет следующий прототип:

```
BOOL TextOut (
        HDC hdc,          // дескриптор контекста устройства
        int nXStart.       // x-координата стартовой позиции
        int nYStart.       // y-координата стартовой позиции
        LPCTSTR lpString, // указатель на символьную строку
```

int cbString // число символов в строке

);

Функция обеспечивает вывод строки с адресом lpString, размещая текст в заданной позиции с учетом текущего режима выравнивания. При выводе используются текущие значения атрибутов контекста устройства – шрифт, цвет текста и цвет фона графических элементов, режим смешивания фона и многие другие.

Функция не распознает конец строки lpString по завершающему нулевому символу, поэтому количество выводимых символов задается параметром cbString.

Задание к лабораторной работе:

1. Необходимо разработать windows-приложение с возможностью построения графика функции в соответствии с вариантом (таблица 8.1).
2. В приложении предусмотреть ввод функции для построения графика, диапазона изменения аргумента.
3. В основном окне приложения строится график введенной функции.
4. Табулирование функции провести с использованием библиотеки, разработанный в предыдущей лабораторной работе.

Таблица 8.1

№ вар.	Функция	x_n	x_k	№ вар.	Функция	x_n	x_k
1.	$y = \frac{arctg(x)}{1+\sin^2 x}$	2	5	2.	$y = \frac{1+\sqrt{0{,}5x}}{0{,}5+\sin^2 x}$	2	4
3.	$y = \ln\left(x^2+2x+2\right)$	-3	0	4.	$y = e^x \sin x \cos^3 x$	-1	1
5.	$y = \frac{\sin^2 x}{\sqrt{x}+x}$	2	5	6.	$y = \frac{e^{0,1x}+1}{1+\cos^2 x}$	0	3
7.	$y = 2x^3 - 6x^2 - 18x + 7$	-2	2	8.	$y = 2x^3 - 3x^2$	1	2
9.	$y = x\sqrt{1+x^2}\cdot\sin x$	1	5	10.	$y = -\ln\left(x^2-4x+5\right)$	1	4
11.	$y = e^x\left(\sin 3x - 3\cos 3x\right)$	1	4	12.	$y = \left(x^2-2x\right)\ln x - \frac{3}{2}x^2 + 4x$	1	4
13.	$y = \frac{\sin^3 x}{x+2}$	0	3	14.	$y = \frac{3x^2+4x+4}{x^2+x+1}$	-1	1

№ вар.	Функция	x_n	x_k	№ вар.	Функция	x_n	x_k
15.	$y=\frac{(1+2x)^2}{1.8+\cos^3 x}$	2	4	16.	$y=\frac{1}{\ln(x^4+4x^3+30)}$	-1	3
17.	$y=x\sin x+\cos x-\frac{1}{4}x^2$	-1,5	1,5	18.	$y=3\cos^2 x-\cos^3 x$	1	5
19.	$y=\frac{1-x+x}{1+x+x^2}$	-2	0	20.	$y=\frac{0,5+\sin^2 x}{-0,5}$	0	4
21.	$y=\frac{\sin^2 x-0,5}{2x}$	1	4	22.	$y=ctg\left(\sqrt[3]{1+x^2}\right)$	-2	2
23.	$y=2\sin x+\cos 2x$	-3	0	24.	$y=xe^x(\cos x+\sin x)$	-3	0
25.	$y=\frac{5-\sqrt{x}}{1+\cos^2 x}$	1	3	26.	$y=3\sin^2 x-\sin^3 x$	3	7
27.	$y=\sin\left(\sqrt{1+x^2}\right)$	-2	2	28.	$y=x\sin xarctgx$	0	4
29.	$y=\frac{1}{2}(x^2+1)arctgx-\frac{\pi}{8}x^2-$	-1	3	30.	$y=3\sin 3x$	0	2

Требования по оформлению отчета:

Отчет должен включать следующие пункты:

1. Титульный лист.
2. Цель работы.
3. Задание.
4. Листинг программного кода.
5. Результаты выполнения программы.
6. Выводы.

Контрольны вопросы для защиты:

1. Что такое GDI?
2. С какими устройствами работает библиотека GDI?
3. Какие базовые примитивы используются для вывода графики?
4. Как вывести текст?
5. Как нарисовать линию?
6. Как изменить цвет, толщину и вид пера для прорисовки?
7. Как нарисовать кубический полином?

Лабораторная работа № 9
Получение информации о процессоре на основе команды CPUID

Цель работы: изучить принципы получения информации о характеристиках конкретного процессора с использованием встроенной уоманды CPUID.

Краткие теоретические сведения:

Для получения информации о процессоре в языке Assembler существует специальная команда CPUID. Она позволяет получить как версию процессора, так и поддерживаемые им возможности. Перед выполнением команды CPUID в регистр EAX следует записать определенное числовое значение, от которого будет зависеть формат возвращаемых данных. Например, для процессоров Intel формат данных показан в таблице 9.1.

Таблица 9.1 – Формат данных для процессора Intel

Значение регистра EAX	**Тип получаемой информации**
00h	Регистр EAX: максимальное входное значение Регистр EBX: 756E6547h (“uneG”) Регистр ECX: 49656E69h (“leni”) Регистр EDX: 6C65746Eh (“letn”)
01h	Регистр EAX: версия (модель, тип, семейство) Регистр EBX: зарезервирован Регистр ECX: зарезервирован Регистр EDX: информация о дополнительных свойствах
02ħ	Регистр EAX: информация о встроенном кэше Регистр EBX: информация о встроенном кэше Регистр ECX: информация о встроенном кэше Регистр EDX: информация о встроенном кэше

Если входное числовое значение равно 00h, в регистр EAX записывается максимальное значение, которое может использоваться (в EAX) с командой CPUID (в данном случае 2). Регистры EBX, ECX

и EDX содержат составляющие имени производителя в ASCII-кодах (12 символов), причем первое значение всегда расположено в младшем регистре (BL, CL и DL). Для процессоров фирмы AMD используется строка «AuthenticAMD”, а для Cyrix — «CyrixInstead».

Если числовое значение равно 0lh, в регистр EAX записывается информация о версии процессора. Формат этого регистра показан в таблице 9.2. Регистры EBX и ECX зарезервированы и не используются. В регистр EDX записывается информация о дополнительных свойствах процессора. Формат этого регистра представлен в таблице 9.3.

Входное числовое значение 02h позволяет получить информацию о кэшах. В таблице 9.4 перечислены некоторые значения для определения размера внутреннего кэша процессора. При этом в регистр EAX записывается количество вызовов CPUID (02h), необходимое для получения всей информации.

Таблица 9.2 – Формат регистра с версией CPU

Биты	**31-14**	**13-12**	**11-8**	**7-4**	**3-0**
Описание	Резерв	Тип	Family ID	Model ID	Stepping ID

Биты 3-0 определяют модификацию процессора. Для Intel это значение будет больше 3, если установленный процессор выше 80486.

Биты 7-4 определяют модель процессора. Значение этого поля зависит от соседнего поля Family ID и применяется совместно с ним, чтобы идентифицировать процессор.

Биты 11-8 определяют номер семейства процессора.

Биты 13-12 для старых процессоров указывают один из следующих типов: 00b – OEM, 01b – Overdrive, 10b – Dual.

Биты 31-14 зарезервированы и не используются (должны быть равны 0).

Таблица 9.3 – Формат регистра свойств процессора

Бит	**Обозначение**	**Описание**
0	FPU	Наличие математического сопроцессора
1	VME	Поддержка виртуального режима V86

Бит	Обозначение	Описание
2	OE	Поддержка точки останова для отладчика
3	PSE	Поддержка расширенных размеров страниц (до 4 Мбайт)
4	TSC	Поддержка команды RDTSC и CR4
5	MSR	Поддержка команд RDMSR и WRMSR
6	PAE	Поддержка расширенных адресов (более 32 бит)
7	MCE	Поддержка проверки машинных ошибок
8	CX8	Поддержка команды CMPXCHG8B
9	APIC	Поддержка встроенного контроллера прерываний APIC
10	-	Резерв
11	SEP	Поддержка быстрых системных вызовов (команды SYSENTER И SYSEXIT)
12	MTRR	Поддержка регистров диапазона памяти MTRR
13	PGE	Поддержка PGE в CR4
14	MCA	Поддержка проверки машинной архитектуры
15	CMOV	Поддержка расширенных команд CMOV
16	PAT	Поддержка таблицы атрибутов, позволяющей увеличить адресуемvю память
17	PSE-36	Поддержка 36 разрядных расширенных страниц (страницы размером 4 Мбайт позволяют использовать физическую адресацию памяти свыше 4 Гбайт)
18-22	-	Резерв
23	MMX	Поддержка технологии MMX
24	FXSR	Поддержка быстрых команд для чисел с плавающей точкой (FXSAVE и FXRST0R)
25-31	-	Зарезервированы и должны быть равны 0

Таблица 9.4 - Значения второго кэша процессора

Код	**Размер кэша, Кбайт**
40h	Кэш отсутствует
41h	128
42h	256
43h	512
44h	1024
45h	2048
79h	128
7Ah	512
7Bh	1024
7Ch	2048
82h	256
83h	512
84h	1024
85h	2048

Кроме стандартных аргументов, процессор может поддерживать расширенные значения для команды CPUID, перечисленные в таблице 9.5. Эти значения следует использовать для получения данных о совместимых с Intel процессоров AMD и Cyrix.

Таблица 9.5 – Расширенные значения для команды CPUID

Код	**Описание**
03h	Позволяет получить серийный номер процессора (начиная с Intel Pentium III), закодированный в регистрах ECX и EDX)
80000000h	Максимальное расширенное значение, поддерживаемое гіроцессором
80000001h	Возвращает версию и свойства для процессора
80000002h	Название процессора
80000003h	Название процессора

Код	Описание
80000004h	Название процессора
80000005h	Возвращает информацию о размерах первого кэша
80000006h	Возвращает информацию о размерах второго кэша

Задание к лабораторной работе:

Разработать Windows-приложение с использованием WinAPI, получающее информацию о процессоре с использованием команды CPUID.

Требования по оформлению отчета:

Отчет должен включать следующие пункты:

1. Титульный лист.
2. Цель работы.
3. Задание.
4. Листинг программного кода.
5. Результаты работы программы.
6. Выводы.

Контрольные вопросы для защиты:

1. Как получить информацию о производителе процессора?
2. Как получить серийный номер процессора?
3. Как получить информацию о КЭШ памяти?
4. Как получить информацию о свойствах процессора?

Лабораторная работа № 10
Мониторинг системы с использованием WinAPI

Цель работы: изучить основные функции WinAPI для получения информации о характеристиках вычислительной машины.

Краткие теоретические сведения:

Функция GetSystemInfo

Функция

void WINAPI GetSystemInfo(_Out_ LPSYSTEM_INFO lpSystemInfo)

принимает в качестве параметра указатель на структуру SYSTEM_INFO

```
typedef struct _SYSTEM_INFO {
 union {
  DWORD  dwOemId;
  struct {
   WORD wProcessorArchitecture;
   WORD wReserved;
  };
 };
 DWORD     dwPageSize;
 LPVOID    lpMinimumApplicationAddress;
 LPVOID    lpMaximumApplicationAddress;
 DWORD_PTR dwActiveProcessorMask;
 DWORD     dwNumberOfProcessors;
 DWORD     dwProcessorType;
 DWORD     dwAllocationGranularity;
 WORD      wProcessorLevel;
 WORD      wProcessorRevision;
```

} SYSTEM_INFO;

Структура SYSTEM_INFO содержит информацию о системе. Она включает архитектуру и тип процессора, число процессоров в системе, размер страницы памяти и другую информацию.

Поля-члены

dwOemId

устаревший элемент, который сохраняется для совместимости.

wProcessorArchitecture

архитектура процессора, установленного в системе. Это поле может принимать следующие значенияs.

Значение	Описание
PROCESSOR_ARCHITECTURE_AMD64 9	x64 (AMD or Intel)
PROCESSOR_ARCHITECTURE_ARM 5	ARM
PROCESSOR_ARCHITECTURE_IA64 6	Intel Itanium-based
PROCESSOR_ARCHITECTURE_INTEL 0	x86
PROCESSOR_ARCHITECTURE_UNKNOWN 0xffff	Unknown architecture.

wReserved

это поле зарезервировано на будущее.

dwPageSize

размер страницы виртуальной памяти.

lpMinimumApplicationAddress

указатель на наименьший адрес памяти, доступный для приложения.

lpMaximumApplicationAddress

Указатель на наибольший адрес памяти, доступный для приложения.

dwActiveProcessorMask

Маска, отображающая множество процессоров, сконфигурированых в системе. Бит 0 - процессор 0; бит 31 - процессор 31.

dwNumberOfProcessors

количество логических процессоров. Чтобы получить более подробную информацию используется функция **GetLogicalProcessorInformation** . Для получения информации о физических процессорах, общих логических процессорах, используется функция **GetLogicalProcessorInformationEx**.

dwProcessorType

устаревший элемент, сохраняется для совместимости. В настоящее время используются поля wProcessorArchitecture, wProcessorLevel, и wProcessorRevision для определения процессора..

PROCESSOR_INTEL_386 (386)
PROCESSOR_INTEL_486 (486)
PROCESSOR_INTEL_PENTIUM (586)
PROCESSOR_INTEL_IA64 (2200)
PROCESSOR_AMD_X8664 (8664)
PROCESSOR_ARM (Reserved)

dwAllocationGranularity

адрес сегмента, начиная с которого может бать выделена виртуальная память. Для более подробной информации и управлением виртуальной памяти используется функция **VirtualAlloc**.

wProcessorLevel

зависит от уровня архитектуры процессора уровня, используется только для отображения. Для того, чтобы определить доступный набор функций процессора, используется функция **IsProcessorFeaturePresent**. Для значения wProcessorArchitecture - PROCESSOR_ARCHITECTURE_INTEL, wProcessorLevel определяет CPU vendor. Для значения wProcessorArchitecture -

PROCESSOR_ARCHITECTURE_IA64, wProcessorLevel установлен в 1.

wProcessorRevision

указывает версию процессора.

Функия GetLogicalProcessorInformation

Функия

```
BOOL WINAPI GetLogicalProcessorInformation(
  _Out_   PSYSTEM_LOGICAL_PROCESSOR_INFORMATION Buffer,
  _Inout_ PDWORD                                ReturnLength
);
```

Предоставляет информацию о логических процессорах вычислительной системы и разделяемом КЭШе.

Параметры

Buffer [out] – указатель на массив структур SYSTEM_LOGICAL_PROCESSOR_INFORMATION.

ReturnLength [in, out] – размер буфера.

Возвращаемое значение

Функция возвращает TRUE, если хотя бы одна стуктура SYSTEM_LOGICAL_PROCESSOR_INFORMATION записана в buffer. В противном случае возвращается FALSE.

Функция GetLogicalDrives

Функция GetLogicalDrives возвращает число-битовую маску в которой хранятся все доступные диски.

```
DWORD GetLogicalDrives(VOID);
```

Параметры:

Эта функция не имеет параметров.

Возвращаемое значение:

Если функция вызвана правильно, то она возвращает число-битовую маску в которой хранятся все доступные диски (если 0 бит равен 1, то диск "A:" присутствует, и т.д.). Если функция вызвана не правильно, то она возвращает 0.

Функция GetDriveType

Функция GetDriveType возвращает тип диска (removable, fixed, CD-ROM, RAM disk, или network drive).

UINT GetDriveType(LPCTSTR lpRootPathName);

Параметры:

lpRootPathName [in] - указатель на не нулевую стоку в которой хранится имя главной директории на диске. Обратный слэш должен присутствовать! Если lpRootPathName равно NULL, то функция использует текущую директорию.

Возвращаемое значение:

Функция возвращает тип диска. Могут быть следующие значения:

Значение	Описание
DRIVE_UNKNOWN	Не известный тип.
DRIVE_NO_ROOT_DIR	Не правильный путь.
DRIVE_REMOVABLE	Съёмный диск.
DRIVE_FIXED	Фиксированный диск.
DRIVE_REMOTE	Удалённый или network диск.
DRIVE_CDROM	CD-ROM диск.
DRIVE_RAMDISK	RAM диск.

Функция GetVolumeInformation

Функция GetVolumeInformation возвращает информацию о файловой системе и дисках(директориях).

```
BOOL GetVolumeInformation(
    LPCTSTR lpRootPathName,          // имя диска(директории)          [in]
    LPTSTR lpVolumeNameBuffer,       // название диска                 [out]
    DWORD nVolumeNameSize,           // длина буфера названия диска    [in]
    LPDWORD lpVolumeSerialNumber,    // сериальный номер диска         [out]
    LPDWORD lpMaximumComponentLength, // максимальная длина фыйла      [out]
    LPDWORD lpFileSystemFlags,       // опции файловой системы         [out]
    LPTSTR lpFileSystemNameBuffer,   // имя файловой системы           [out]
    DWORD nFileSystemNameSize        // длина буфера имени файл. сист. [in]
);
```

Возвращаемое значение:

Если функция вызвана правильно, то она возвращает не нулевое значение (TRUE). Если функция вызвана не правильно, то она возвращает 0 (FALSE).

Функция GetDiskFreeSpaceEx

Функция GetDiskFreeSpaceEx выдаёт информацию о доступном месте на диске.

```
BOOL GetDiskFreeSpaceEx(
    LPCTSTR lpDirectoryName,            // имя диска(директории)          [in]
    PULARGE_INTEGER lpFreeBytesAvailable, // доступно для использования(байт)
[out]
    PULARGE_INTEGER lpTotalNumberOfBytes,  // максимальный объём( в байтах )
[out]
    PULARGE_INTEGER lpTotalNumberOfFreeBytes // свободно на диске( в байтах )
[out]
);
```

Возвращаемое значение:

Если функция вызвана правильно, то она возвращает не нулевое значение(TRUE). Если функция вызвана не правильно, то она возвращает 0 (FALSE).

Функция GlobalMemoryStatus

Функция GlobalMemoryStatus возвращает информацию о используемой системой памяти.

```
VOID GlobalMemoryStatus(
        LPMEMORYSTATUS lpBuffer   // указатель на структуру
MEMORYSTATUS
);
```

Описание структуры MEMORYSTATUS

```
typedef struct _MEMORYSTATUS {
    DWORD dwLength;             // длина структуры в байтах
    DWORD dwMemoryLoad;         // загрузка памяти в процентах
```

```
    SIZE_T dwTotalPhys;        // максимальное количество физической памяти в
байтах
    SIZE_T dwAvailPhys;        // свободное количество физической памяти в
байтах
    SIZE_T dwTotalPageFile;    // макс. кол. памяти для программ в байтах
    SIZE_T dwAvailPageFile;    // свободное кол. памяти для программ в байтах
    SIZE_T dwTotalVirtual;     // максимальное количество виртуальной памяти в
байтах
    SIZE_T dwAvailVirtual;     // свободное количество виртуальной памяти
в байтах
} MEMORYSTATUS, *LPMEMORYSTATUS;
```

Системная информация

Функции GetComputerName, GetUserNameA

Функция GetComputerName возвращает NetBIOS имя локального компьютера.

```
BOOL GetComputerName(
    LPTSTR lpBuffer,    // имя локального компьютера( длина буфера равна
MAX_COMPUTERNAME_LENGTH + 1 ) [out]
    LPDWORD lpnSize     // размер буфера ( лучше поставить
MAX_COMPUTERNAME_LENGTH + 1 )    [out/in]
    );
```

Функция GetUserName возвращает имя текущего пользователя.

```
BOOL GetUserName(
  LPTSTR lpBuffer,  // имя юзера( длина буфера равна UNLEN + 1 )   [out]
  LPDWORD nSize     // размер буфера ( лучше поставить UNLEN + 1 ) [out/in]
    );
```

Возвращаемые значения:

Если функции вызваны правильно, то они возвращают не нулевое значение(TRUE). Если функции вызваны не правильно, то они возвращают 0 (FALSE).

Функции GetSystemDirectory, GetTempPath, GetWindowsDirectory, GetCurrentDirectory

Функция GetSystemDirectory возвращает путь к системной директории.

```
UINT GetSystemDirectory(
    LPTSTR lpBuffer,  // буфер для системной директории [out]
    UINT uSize        // размер буфера                 [in]
);
```

Эта функция возвращает размер буфера для системной директории не включая нулевогозначения в конце, если она вызвана правильно. Если функция вызвана не правильно, то она возвращает 0.

Функция **GetTempPath** возвращает путь к директории, отведённой для временных файлов.

```
DWORD GetTempPath(
    DWORD nBufferLength,  // размер буфера                 [in]
    LPTSTR lpBuffer       // буфер для временной директории [out]
);
```

Эта функция возвращает размер буфера для системной директории не включая нулевого значения в конце, если она вызвана правильно. Если функция вызвана не правильно, то она возвращает 0.

Функция **GetWindowsDirectory** возвращает путь к Windows директории.

```
UINT GetWindowsDirectory(
    LPTSTR lpBuffer,  // буфер для Windows директории [out]
    UINT uSize        // размер буфера               [in]
);
```

Эта функция возвращает размер буфера для системной директории не включая нулевого значения в конце, если она вызвана правильно. Если функция вызвана не правильно, то она возвращает 0.

Функция GetCurrentDirectory возвращает путь к текущей директории.

```
DWORD GetCurrentDirectory(
    DWORD nBufferLength,  // размер буфера                    [in]
    LPTSTR lpBuffer       // буфер для текущей директории [out]
);
```

Эта функция возвращает размер буфера для системной директории не включая нулевого значения в конце, если она вызвана правильно. Если функция вызвана не правильно, то она возвращает 0.

Задание к лабораторной работе:

Разработать приложение, получающее информацию о системе, используя функции WinAPI. Отобразить информацию об используемых процессорах, оперативной памяти, виртуальной памяти, внешней памяти, параметрах монитора.

Контрольные вопросы для защиты:

1. Что из себя представляет система линейных алгебраических уравнений?
2. Какие численные методы используются для вычисления СЛАУ?
3. В чем суть итерационных методов решения СЛАУ?
4. В чем суть прямых методов решения СЛАУ?
5. Что выполняется на обратном ходе метода Гаусса?
6. Что вычисляется на первом этапе метода прогонки?
7. В чем отличие метода Зейделя от метода последовательных приближений?
8. Как определить условие сходимости итерационного процесса?
9. Как определить условие окончания итерационного процесса?

Список использованных источников

1. Степанов А. Н. Архитектура вычислительных систем и компьютерных сетей : учеб. пособие для вузов. - Санкт-Петербург : Питер, 2007. - 508 с.
2. Таненбаум, Э. Архитектура компьютера / Э. Таненбаум, Т. Остин ; [перевел с англ. Е. Матвеев]. - 6-е изд.. - Санкт-Петербург [и др.] : Питер, 2014. - 811 с.
3. Юров, В. И. Assembler : учебное пособие для вузов / В. И. Юров. - 2-е изд.. - Санкт-Петербург [и др.] : Питер, 2007. - 636 с.
4. Юров, В. И. Assembler : практикум / В. И. Юров. - 2-е изд.. - Санкт-Петербург [и др.] : Питер, 2007. - 398 с.
5. Щупак Ю. А. Win 32 API. Разработка приложений для Windows. - Санкт-Петербург : Питер, 2008. - 592 с.

Самовендюк Николай Владимирович

АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Практикум
по выполнению лабораторных работ для студентов специальности 1-40 04 01 «Информатика и технологии программирования» дневной формы обучения

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 27.01.20.

Рег. № 31Е.
http://www.gstu.by