МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Белорусский национальный технический университет

Кафедра «Робототехнические системы»

Ю. Н. Матрунчик

МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ УПРАВЛЕНИЯ

Лабораторный практикум
для студентов специальности 1-53 01 01 «Автоматизация технологических процессов и производств (по направлениям)»

Минск
БНТУ
2020

УДК 004.31–181.1–022.52 (076.5)
ББК 32.973.26–04Я7
М34

Рецензенты:
заведующий лабораторией робототехнических систем
ОИПИ НАН Беларуси, канд. техн. наук, доцент *Г. А. Прокопович*
заведующий кафедрой «Электрооборудование
сельскохозяйственных предприятий» БГАТУ, канд. техн. наук,
доцент *В. А. Дайнеко*

Матрунчик, Ю. Н.
Микропроцессорные системы управления : лабораторный практикум для студентов специальности 1-53 01 01 «Автоматизация технологических процессов и производств (по направлениям)» / Ю. Н. Матрунчик. – Минск : БНТУ, 2020. – 65 с.
ISBN 978-985-583-138-0.

Учебное пособие представляет собой руководство к выполнению лабораторных работ по дисциплине «Проектирование микропроцессорных систем управления». Каждая тема содержит вопросы и задачи с пояснительным материалом.

УДК 004.31–181.1–022.52 (076.5)
ББК 32.973.26–04Я7

ISBN 978-985-583-138-0

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Появление микропроцессорной техники (микропроцессоров, микропроцессорных комплектов и секций, микроЭВМ и микроконтроллеров) было обусловлено, прежде всего, потребностью автоматизации технологических процессов и производств, так как для этой цели требовались высоконадежные средства вычислительной техники.

Проблема повышения надежности ЭВМ встала особенно остро после первых попыток внедрения больших ЭВМ второго поколения для комплексной автоматизации (контроль, сигнализация и отображение информации, реализация алгоритмов логики и дискретного управления, цифровое непосредственное управление). Главная причина неудачных внедрений ЭВМ второго поколения – их низкая надежность. В дальнейшем опыт показал, что легче поддаются внедрению системы централизованного контроля, децентрализованные системы управления на базе специализированных цифровых вычислительных устройств, локальные информационно-управляющие системы. Другое направление автоматизации технологических процессов связано с использованием управляющих вычислительных комплексов на базе более надежных транзисторных миниЭВМ, а затем выполненных на основе микросхем низкой степени интеграции.

Ни одно из направлений, сложившихся к концу 70-х годов, не позволяло перейти к комплексной автоматизации технологических процессов. Для решения этой проблемной задачи требовалось значительно повысить надежность элементной основы ЭВМ.

Революционным прорывом принято считать 1971 год, когда американская фирма INTEL разработала первый микропроцессор, ознаменовавший новый период в развитии вычислительной техники и систем управления на их основе. Уже в 1973 году фирмой Control Logic была разработана первая микроЭВМ для целей управления, в 1974 году была создана первая микроЭВМ с микропрограммным управлением, в 1975 году – первая персональная микроЭВМ, получила широкое признание концепция полностью распределенных систем управления, локальных вычислительных сетей и интегрированных компьютерных производств. Бурное развитие микропроцессорной техники и систем на ее основе стало возможным благодаря ее основным особенностям: высокой надежности, относительно не-

высокой стоимости и малым габаритам. Использование микропроцессорной техники в системах управления позволило получить целый ряд дополнительных преимуществ, реализовать не только цифровые микропроцессорные регуляторы, но и адаптивные регуляторы, реализующие оптимальные алгоритмы управления.

Внедрение микропроцессорных систем управления потребовало от специалистов дополнительных аппаратных и программных решений, таких как: разработку устройств сопряжения с объектом, выбор и реализацию метода гальванической развязки, разделение сигнальных и силовых проводников, экранирование, аналоговую фильтрацию, выбор и обоснование метода цифровой фильтрации, разработку алгоритмов управления и др.

Кроме того, использование микропроцессорной техники в системах управления требует от разработчиков дополнительных знаний как об аппаратных, так и о программных особенностях, связанных с управлением внешними устройствами и устройствами пользователя, что необязательно знать обычному пользователю при решении математических задач или работая в режиме использования специализированных и универсальных пакетов.

В данном методическом пособии описываются основные принципы микропроцессорного управления внешними устройствами. Предложенные для самостоятельного выполнения задания направлены на закрепление теории и приобретения практических навыков разработки управляющих программ пользователя.

1. ПРИНЦИПЫ УПРАВЛЕНИЯ ВНЕШНИМИ УСТРОЙСТВАМИ МИКРОЭВМ

Любая микропроцессорная система имеет модульный принцип построения, то есть все функциональные блоки выполняются в виде конструктивно законченных устройств. Принцип модульности является одной из современных особенностей архитектурного и структурного построения микроЭВМ. Этот принцип распространяется как на аппаратную (техническую), так и на программную части системы. Модульность предусматривается на всех иерархических уровнях семейства узлов определенного назначения для вычислительных, контролирующих и управляющих систем.

Связь функционально законченных модулей в микропроцессорной системе осуществляется с помощью электрических проводников, называемых линиями. Линии, сгруппированные по некоторому признаку или назначению, объединяются в шины (шина данных ШД, адресная шина АШ и шина управления ШУ). Совокупность шин, служащих для обмена информацией между компонентами системы, образует магистраль.

Периферийные устройства (ПУ) имеют существенно меньшее быстродействие по сравнению с основной памятью микроЭВМ (оперативной, постоянной или комбинации обоих видов). Это позволяет организовать параллельную работу подобных устройств через один и тот же канал за счет разделения их работы во времени, то есть за счет мультиплексирования канала.

Процессор формирует команды ввода-вывода. Канал получает необходимую информацию о команде ввода-вывода от центрального процессора (ЦП), связывается с каналом ввода-вывода (КВВ), выбирает из основной памяти (ОП) команды канала, дешифрирует и выполняет их. КВВ осуществляет логическое подключение к каналу запрашиваемого ПУ, передает приказы в ПУ, преобразовывает форматы данных ПУ в формат стандартного интерфейса ввода-вывода, формирует запросы в ПУ в соответствии с приоритетом ПУ на передачу данных через интерфейс ввода-вывода; управляет последовательностью функционирования ПУ; определяет готовность ПУ для связи с каналом. ПУ выполняет различные приказы КВВ и вспомогательные функции.

Разделение системы ввода-вывода по уровням иерархии для реализации процесса ввода-вывода обеспечивает гибкость программного управления вводом-выводом благодаря возможной параллельной работе ЦП, каналов, КВВ и нескольких ПУ.

В настоящее время используются программируемые микроконтроллеры прямого доступа в память, что способствует значительному повышению быстродействия и эффективности работы микроЭВМ и микропроцессорной системы.

Связь между двумя устройствами, подключенными к каналу, осуществляется по принципу "управляющий – управляемый" (активный – пассивный).

В любой момент времени только одно устройство является активным. Активное устройство управляет циклами обращения к каналу, удовлетворяет (если это необходимо) требования прерывания от внешних устройств, контролирует предоставление прямого доступа к памяти. Пассивное устройство (управляемое) является только исполнительным устройством. Оно может принимать или передавать информацию только под управлением активного устройства. Типичным примером таких отношений является центральный процессор (как активное устройство), выбирающий команду из памяти, которая всегда является пассивным устройством. Другим примером активного устройства может служить устройство, работающее в режиме прямого доступа к памяти, при этом память является пассивным устройством.

Связь через канал является замкнутой, то есть на управляющий сигнал, передаваемый активным устройством, должен поступить ответный сигнал от пассивного устройства. Время ответа не должно превышать максимально допустимого.

Асинхронное выполнение операций передачи данных устраняет необходимость в тактовых импульсах. В результате этого обмен с каждым устройством может происходить с максимально возможным для данного устройства быстродействием.

Канал обеспечивает три типа обмена данными, а именно: программный обмен; обмен в режиме прямого доступа к памяти (ПДП) и обмен в режиме прерывания программ.

Программный обмен данными – это передача данных по инициативе и под управлением программы. Обычно перед началом об-

мена проверяется содержимое **регистра состояния** устройства, чтобы определить, готово ли оно к обмену данными.

Обмен в режиме прямого доступа к памяти (ПДП) является самым быстрым способом передачи данных между памятью и внешним устройством. Такой обмен осуществляется по инициативе внешнего устройства и не изменяет состояние центрального процессора. Поэтому он может выполняться в промежутках между циклами обращения к каналу, проводимых центральным процессором.

Адресация и управление размерами передаваемого массива данных находятся под управлением устройства, получившего прямой доступ к памяти. Массивы данных ПДП могут передаваться со скоростью, определяемой быстродействием памяти.

Обмен данными в режиме прерывания программы – этот режим осуществляется по требованию внешних устройств. В этом случае ЦП приостанавливает выполнение текущей программы, чтобы обслужить запрашивающее устройство. После завершения выполнения программы обслуживания ЦП возобновляет выполнение прерванной программы с того места, где она была прервана.

2. ПРЕДСТАВЛЕНИЕ ЧИСЕЛ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ. РАЗРЯДНЫЕ СЕТКИ МИКРОЭВМ

Общепринятый формат двоичного слова для данного типа любой ЭВМ, принято называть разрядной сеткой.

В двоичной системе любое десятичное число может быть представлено соответствующей последовательностью двоичных цифр:

$$X_{10} = a_{k-1}a_{k-2}\ldots a_1a_0,\ a_{-1}a_{-2}\ldots ,$$

где $a_i = 0$ или 1;

k – количество цифр в целой части числа.

Эта запись соответствует коэффициентам многочлена, которым представляется любое число десятичной системы при разложении в ряд с основанием 2.

В качестве основной используется форма представления чисел с плавающей запятой, которая практически не требует масштабирования.

В общем виде любое число X с плавающей запятой имеет вид:

$$X = \pm q \cdot S^{\pm p}, \qquad (2.1)$$

где q – мантисса числа X;

S – основание характеристики, которое совпадает обычно с основанием системы счисления;

p – порядок.

Для двоичной системы счисления мантисса q – это правильная двоичная дробь. Поэтому

$$X = \pm q \cdot 2^{\pm p}, \qquad (2.2)$$

где q и p – двоичные числа, причем $|q| < 1$.

В разрядной сетке старший разряд отводится под знак мантиссы, затем следует разряд знака порядка, следующие 7 разрядов предназначены для размещения значения порядка p, оставшиеся 7 разрядов – для размещения старших разрядов мантиссы q.

Во второй части разрядной сетки размещаются младшие разряды мантиссы. Веса разрядов, в которых размещаются порядок и мантисса, показаны на рис. 2.1. Из рисунка видно, что под мантиссу отведено 23 разряда, причем сразу за нулевым разрядом порядка p_0 следует разряд мантиссы q_{-2} с весом 2^{-2}, а старший разряд мантиссы q_{-1} с весом 2^{-1} отсутствует в реальной разрядной сетке. И это действительно так. Принято считать, что старший разряд мантиссы находится в **скрытом** разряде и всегда равен "1".

Рис. 2.1. Размещение вещественного числа в разрядной сетке

Это обстоятельство, пожалуй, самое сложное для понимания, так как, с одной стороны, разряд отсутствует, как на рис. 2.1, так и при выводе числа из ОЗУ на экран, а с другой стороны, его необходимо учитывать при определении истинного значения мантиссы при вычислениях.

Такое решение обусловлено чисто техническими (аппаратными) трудностями. Известно, что для представления каждого разряда числа в 8 с/с требуется три двоичных разряда, а число 23 под мантиссу не кратно трем. Если уменьшить количество разрядов под порядок, то сужается диапазон представления чисел с плавающей запятой. Поэтому недостачу старшего разряда мантиссы решено компенсировать косвенным путем.

Нормализованным двоичным числом называется такое двоичное число, для которого выполняется следующее условие:

$$1/2 \le |q| < 1$$

Это значит, что старший разряд мантиссы **нормализованного** двоичного числа всегда равен "1".

Например, двоичное число $0.00101..0*2^{0...011}$ после нормализации имеет вид:

$$0.\underline{1}01...0*2^{0...01}$$

При нормализации двоичного числа производится сдвиг мантиссы влево, а порядок уменьшается на количество сдвигов. Для рассмотренного выше примера мантисса в разрядной сетке имеет другой вид: 0.01...0, а старший разряд с весом 2^{-1} подразумевается как скрытый (скрытый разряд в примере подчеркнут).

Нормализация чисел выполняется в микроЭВМ автоматически, результат вычислений также подвергается нормализации.

Таким образом, очевидно, что при использовании чисел с плавающей запятой арифметическое устройство помимо аппаратуры, выполняющей операции над мантиссами, должно иметь аппаратуру для операций над порядками и для вспомогательных операций (выравнивание порядков, нормализация и др.).

Для упрощения операций над порядками их сводят к действиям над целыми положительными числами, используя представление чисел с плавающей запятой со смещенным порядком.

Смещенный порядок p^* получается путем прибавления к порядку р целого положительного числа К, называемого смещением:

$$p^* = p + K, \tag{2.3}$$

где $\mathrm{K} = |p_{max}| + 1 = 2^m$;

m – количество разрядов, отведенных под порядок (n_p = 7 согласно рис. 2.2);

$|p_{max}|$ = 1111111.

Отсюда следует, что смещение К равно коэффициенту пересчета счетчика, то есть

$$\mathrm{K} = 2^7 = 10000000_2 = 200_8$$

Здесь индексы 2 и 8 обозначают соответствующие системы счисления.

Смещенный порядок p^* всегда положителен. Для его представления нужно количество разрядов, равное сумме разрядов порядка (n_p) и знака порядка, то есть

$$n_p{}^* = n_p + 1 = 8.$$

При $n_p = 7$ значение порядка p с учетом знака изменяется в диапазоне от +127 до –128. Смещенный порядок в этом диапазоне будет выглядеть следующим образом (см. табл. 2.1):

Таблица 2.1

Смещенный порядок

10 с/с	2 с/с	8 с/с	16 с/с
+127	11 111 111	377	FF
...	…	...	…
+1	10 000 001	201	81
0	10 000 000	200	80
-1	01 111 111	177	7F
…	…	...	…
-128	00 000 000	000	00

Таким образом, смещенный порядок изменяется от 0 до 377_8 или FF_{16}.

Для получения смещенного порядка необходимо помнить, что в микроЭВМ системы Intel операции сложения и вычитания заменяются алгебраическим сложением с использованием дополнительного кода.

Правило алгебраического сложения двух двоичных чисел с использованием дополнительного кода: положительные слагаемые представляются в прямом коде, а отрицательные – в дополнительном. Производится арифметическое суммирование этих кодов, включая разряды знаков, которые при этом рассматриваются как разряды целых единиц. При возникновении переноса из знакового разряда единица циклического переноса игнорируется. В результате получается алгебраическая сумма в прямом коде, если эта сумма положительна, и в дополнительном коде, если эта сумма отрицательна.

Так как К > $|p_{max}|$, то при получении p^* сумма всегда будет положительна, если $p > 0$, причем для получения p^* достаточно в знаковый разряд порядка записать "1". Рассмотрим пример когда $p < 0$.

Пример: Пусть $p = -127$

Решение:
Прямой код числа –127:(ПК–127) = 1 1111111
Обратный код числа –127:(ОК–127) = 1 0000000
Дополнительный код числа –127:(ДК–127) = 1 0000001

$$p^* = \begin{array}{rr} & 10000001 \quad (\text{ДК } -127) \\ + & \hline 10000000 \quad (\text{К}) \\ & 100000001 \quad (\text{результат}) \\ & \boxed{\text{единица циклического перноса}} \end{array}$$

Итак, $p^* = 00000001_2$

Формула (2.3) не дает однозначного перехода от смещенного порядка p^* к действительному p.

Предлагается простой способ нахождения порядка p. При этом учитывается состояние старшего разряда смещенного порядка p^*. Из табл. 2.1 видно, что для всех значений от 0 до +127 в старшем разряде двоичного кода смещенного порядка присутствует "1", а для всех значений отрицательного p – "0". Поэтому для перехода к действительному порядку необходимо выполнить следующее:

– восстановить знак порядка p, проинвертировав старший разряд смещенного порядка p^*;

– считать полученный код результатом вычисления действительного порядка и, используя сформулированное выше правило алгебраического сложения двух двоичных чисел с дополнительным кодом (п. 2.3), записать код порядка p.

Примеры:
Найти порядок p.

1. $p_1^* = 10001010$
2. $p_2^* = 00000001$

Решение: Восстанавливаем знаки порядков:

1. P_1 = **0**0001010 (ПК)
2. P_2 = **1**0000001 (ДК)

Знаковые разряды в P_1 и P_2 выделены. Согласно принятому правилу кодирования знаков порядка число $P_1 > 0$, а $P_2 < 0$. Так как P_1 и P_2 являются результатами вычислений, то, согласно правилу алгебраического сложения с использованием дополнительного кода, P_1 получено в прямом коде (ПК), а P_2 – в дополнительном коде (ДК).

Значит,

p_1 = ПК (P_1),

а p_2 = ДК (P_2).

Отсюда следует, что $p_1 = P_1 = +10_{10}$. Для получения p_2 необходимо найти дополнительный код от P_2. Учитывая, что дополнительный код от дополнительного кода числа дает прямой код, получим p_2:

ОК (P_2) = 11111110

ДК (P_2) = $11111111_2 = -127_{10}$

В табл. 2.2 показано представление некоторых характерных чисел с плавающей запятой: теоретическое (со всеми разрядами) и в памяти микроЭВМ (с использованием скрытого разряда). В теоретическом представлении числа записаны в одну строку, в которой предполагается 32 разряда, при этом смещенный порядок выделен, а старший разряд мантиссы, который становится скрытым в ОЗУ микроЭВМ, подчеркнут.

Следует обратить внимание на следующее:

– положительный знак числа ("0") при выводе на экран видеотерминала и печатающее устройство в 2 с/с не отображается;

– код числа в 8 с/с и 16 с/с выводится на экран, печатающее устройство без учета скрытого разряда;

– код двоичного числа, хранимого в памяти микроЭВМ, выводится в 10 с/с с учетом скрытого разряда;

– для перевода в 8 с/с двоичного кода, хранимого в ОЗУ микроЭВМ, сохраняется известное правило: двоичное число разбивается на триады влево и вправо от запятой, которая зафиксирована перед старшим разрядом мантиссы, а триады представляются восьмеричными эквивалентами;

– для перевода в 16 с/с двоичного кода, хранимого в ОЗУ микроЭВМ, двоичное число разбивается на тетрады влево и вправо от

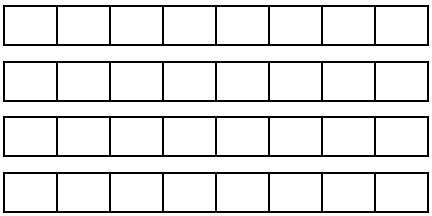
запятой, которая зафиксирована перед старшим разрядом мантиссы, а тетрады представляются шестнадцатеричными эквивалентами.

Таблица 2.2

Представление чисел в ЭВМ

Исходные данные			Нормализ.	Представление чисел в формате 4-х слов			
10 с/с	8 с/с	16 с/с	2 с/с	2 с/с		на экране	
				Теоретическое	в ОЗУ микроЭВМ	8 с / с	16 с / с
1	2	3	4	5	6	7	8
4	4	4	0.100*23	0**10000011** 100...0	10000011__ 00...0	203_0_0_0	83_0_0_0
2	2	2	0.100*22	0**10000010** 100...0	10000010__ 00...0	202_0_0_0	82_0_0_0
1	1	1	0.100*21	0**10000001** 100...0	10000001__ 00.....0	201_0_0_0	81_0_0_0
0.5	0.4	0.8	0.100*20	0**10000000** 100...0	10000000__ 00...0	200_0_0_0	80_0_0_0
0.25	0.2	0.4	0.100*2-1	0**01111111** 100...0	01111111__ 00...0	177_0_0_0	7F_0_0_0
0.125	0.1	0.2	0.100*2-2	0**01111110** 100...0	01111110__ 00....0	176_0_0_0	7E_0_0_0
0.0625	0.01	0.1	0.100*2-3	0**01111101** 100...0	01111101__ 00...0	175_0_0_0	7D_0_0_0
-4	-4	-4	0.100*23	1**10000011** 100...0	10000011__ 10... 0	203__20_0	83_80_0_0
–0.0625	–0.01	–0.1	0.100*2-3	1**01111101** 100...0	01111101__ 00...0	175__20_0	7D_80_0_0

В микроЭВМ INTEL ввод-вывод осуществляется по байтам. В старшем байте мантиссы (см. рис. 2.1) всего 7 разрядов, поэтому знак q переносится в этот байт. Тогда структура 4-х байтов в памяти выглядит, как показано на рис. 2.2.



– смещенный порядок p^*

– старший байт q со знаком

– средний байт q

– младший байт q

Рис. 2.2. Структура 4-х байтов при вводе/выводе

Задание:

1. Перевести число из 10 с/с в 2 с/с, 8 с/с и 16 с/с, разместить его в 32-разрядную сетку.

2. Перевести двузначное число в 10 с/с, написать программу записи и считывания числа в ячейки памяти. Запись производить в 8 с/с и 16 с/с. Восстановить смещенный порядок.

Таблица 2.3

Варианты заданий

№	Число А в 10 с/с	Двоичное число В с плавающей запятой	Смещенный порядок Р
1	1275 430 –1275 –430	а) 1010101011010101 1010101010101010 б) 0101010101011010 0101010101010101	а) 01100110 б) 10011011 в) 01110010
2	3040 860 –3040 –860	а) 0000111101110011 1101100010001111 б) 1111000100011111 0001111000010000	а) 10011001 б) 01110100 в) 00001110
3	6020 1720 –6020 –1720	а) 1111000101111001 0001111000110010 б) 0111110000001110 1100111100110011	а) 01010111 б) 10111101 в) 00001111
4	265 3875 –265 –3875	а) 0001110010011100 1111000011110101 б) 1111101101000111 0000110001110100	а) 10011100 б) 00011110 в) 00001011
5	775 4375 –775 –4375	а) 1000001111110000 0001110000110011 б) 0000001111100011 1100000011011111	а) 01111100 б) 10011110 в) 00101101
6	795 1807 –795 –1807	а) 0110011000011111 0001111000111111 б) 1000111000111000 1111000111000111	а) 01110001 б) 11111000 в) 00011100
7	2335 35 –2335 –35	а) 1001100111100000 1110011000000000 б) 0111000111000111 0000111000111000	а) 10001110 б) 00000111 в) 01100111
8	233 4830 –233 –4830	а) 0111100011110000 1111000011110011 б) 1000011000001111 0000111111100000	а) 00011100 б) 00101011 в) 11111100

3. УПРАВЛЕНИЕ ТЕХНОЛОГИЧЕСКИМ ПРОЦЕССОМ (ОБЪЕКТОМ) С ПОМОЩЬЮ МИКРОЭВМ

Задачи, стоящие перед разработчиком микропроцессорной системы управления

Микропроцессорная система управления (МСУ), как правило, содержит две основные части: аппаратную и программную.

Перед разработчиком МСУ стоят следующие довольно сложные задачи:

1. Изучить объект (процесс) управления его особенности и степень его подготовленности к автоматизации.

2. Разработать алгоритм контроля и управления технологическим процессом (объектом) с учетом выбранного или заданного критерия оптимального управления. В общем случае могут подлежать разработке как простые законы дискретного управления (типа "да – нет"), так и более сложные алгоритмы непосредственного цифрового управления, включая сложные законы оптимального управления.

3. Исходя из разработанного алгоритма управления, разделить МСУ на аппаратную и программную части с учетом дополнительных критериев (надежности, стоимости, быстродействия и др.).

4. Выбрать микропроцессорные средства вычислительной техники для проектирования аппаратной части системы (серийный микроконтроллер, микроЭВМ, персональный компьютер, специализированный контроллер и т. д.).

5. При выборе микроЭВМ или персонального компьютера необходимо разработать устройство сопряжения с объектом (УСО).

6. Разработать программное обеспечение (ПО) системы, включающее, кроме операционной системы (ОС) или отдельных файлов ОС, основную управляющую программу (ОУП) пользователя.

При разработке ОУП неизбежно встанет вопрос об организации обмена информацией между микропроцессорным блоком управления и объектом через УСО. На этой стадии необходимо выбрать режим обмена: программный, прямого доступа в память, по прерыванию или комбинированный. Этот выбор во многом определяет эффективность обмена информацией и накладывает отпечаток на степень сложности, а значит, и стоимости ОУП.

Алгоритм контроля и управления может быть опробован при натурных испытаниях, то есть непосредственно на объекте, либо на испытательном стенде с имитационной моделью объекта, либо с помощью моделирующей программы. В последнем случае моделирующая программа должна поддерживать объект. Степень поддержки объекта зависит от задачи, которая стоит перед разработчиком: проверка правильности программы, реализующей алгоритм, или проверка правильности программы и контроля поведения объекта и т. д. Задачи, которые ставит разработчик перед моделирующей программой, определяют ее сложность и стоимость, что дополнительно увеличивает затраты на разработку и испытания МСУ.

При моделировании на микроЭВМ пользователь может использовать свою программу так, как будто существует сам объект.

Однако существует определенная проблема, которая состоит в том, что при отсутствии УСО нарушается принцип асинхронной замкнутой связи, а поэтому адреса регистров и порты ввода-вывода, заложенные в ОУП, не существуют.

Порты ввода-вывода (ПВВ) – это блоки (модули), задачей которых является осуществление взаимодействия между микропроцессорной системой (МПС) и внешней средой (внешними коммуникациями). Таким образом, ПВВ – это унифицированное средство подключения внешних устройств к МПС, т. е. элементы интерфейса.

Порт ввода ($П_{вв}$) – это любой источник данных, например, (но не обязательно) адресуемый регистр, подключенный к шинам МПС. Он выдает слово в МП, когда к нему осуществляется обращение.

Порт вывода ($П_{выв}$) – это любой приемник данных, например, адресуемый регистр, подключенный к шинам МПС. Он получает слово от МП, когда последний обращается к нему.

Таким образом, $П_{вв}$ – это адресуемые одно- или двунаправленные буферные регистры, предназначенные для построения программируемого интерфейса. $П_{вв}$ имеют свои адреса, поэтому к МПС может быть подключено несколько внешних устройств (ВУ).

Каждый порт является составной частью интерфейса между МП и каким-либо ВУ, например, датчиком, ЦАП, АЦП, терминалом, внешней памятью и т. д.

В большинстве МП для адресации портов (т. е. для выборки нужного порта) используется адресная шина или ее часть. Очень часто

адреса $П_{вв}$ отличаются от адресов $П_{выв}$ и от адресов памяти не значениями, а сигналами на соответствующих управляющих линиях.

С каждым внешним устройством обычно связано несколько портов, выполняющих определенные функции на различных стадиях обмена. В одни порты заносятся управляющие признаки и адреса (номера) внутренних регистров обслуживаемой микросхемы. Другие порты используются для обмена данными между центральным процессором и выбранными внутренними регистрами. По содержимому третьих портов операционная система получает информацию о состоянии внешнего устройства, анализирует признаки завершения очередной процедуры обмена, контролирует правильность функционирования оборудования.

Содержимое неподключенного порта соответствует коду 377_8 или FF_{16};

При моделировании проблему отсутствия реальных УСО можно обойти двумя путями, заменив реальные адреса регистров моделирующими адресами:

– выделить область адресов ОЗУ, которая не используется при программировании на том языке, который используется при моделировании;

– зарезервировать адреса ячеек ОЗУ с помощью переменных.

Имеется два типа программно-управляемого ввода-вывода: с условной и безусловной передачей.

При безусловной передаче обмен данными с портом ввода-вывода осуществляется без предварительного определения готовности порта принимать или передавать данные.

Как правило, безусловная передача используется для обмена командной информацией, а также данными о состоянии устройств; иногда применяется при обмене информацией с внешними устройствами. При таком обмене предполагается, что устройство ввода-вывода готово принимать или передавать данные. Ошибки при обмене данными могут возникать в тех случаях, если при программировании не принимается необходимых мер предосторожности. Если микропроцессор направляет данные в более быстром темпе, чем тот, в котором выходной порт в состоянии их принимать, данные будут потеряны. Что касается входного порта, то он будет предоставлять микропроцессору избыточные данные, если обращение последнего к порту будет следовать со слишком большой частотой.

Очевидно, что при таком способе обмена следует очень четко знать аппаратурные особенности микропроцессора, портов, устройства сопряжения с объектом, особенности технологического процесса и т. п.

Чтобы устранить трудности, которые возникают при безусловной передаче информации, применяют условную передачу данных с использованием процедуры квинтирования установления связи. При условной передаче до начала передачи собственно данных осуществляется безусловная передача информации о состоянии из устройства ввода-вывода в процессор. Информация о состоянии выдается в виде комбинации битов, называемой флагом и указывающей на состояние готовности порта ввода-вывода. Необходимость использования подпрограммы или дополнительных операторов для проверки флага состояния вызывает увеличение времени выполнения операции ввода-вывода. Если в системе с программно-управляемым вводом-выводом имеется несколько внешних устройств (узлов), то необходим процесс поочередной проверки флагов всех устройств (узлов). Такой процесс называется опросом флагов.

Флаг состояния, как правило, соответствует состоянию одного из разрядов выбранного для этих целей порта. Этот разряд порта называется флаговым. Если флаг состояния "1", то это свидетельствует о наличии данных. Компьютер принимает данные с входного порта, а сигнал выбора для данного порта разрешает считывание данных с порта и одновременно сбрасывает флаг наличия данных. Аналогично осуществляется выдача управляющей информации во внешние устройства.

Пример:

1) Постановка задачи

Реальный объект управления представляет автоматизированное транспортное средство робототехнического комплекса, управляемое встроенным микроконтроллером. Основная управляющая программа для контроллера после разработки, отладки и апробации заносится в перепрограммируемое постоянное запоминающее устройство (ППЗУ).

Программа обеспечивает возможность движения с различной скоростью (набор переключаемых скоростей $V_1 - V_n$) к различным пунктам назначения, например, к стеллажам (набор переключаемых путей $S_1 - S_n$) и цепь включения напряжения питания – U.

Наборы V_n, S_n и напряжение U представляют узлы управления внешние по отношению к контроллеру, то есть представляют три внешние устройства ВУ. Каждое из этих ВУ имеют регистр данных и флаг состояния. Выбор скорости и пути определяется кодом регистра данных. Регистр данных напряжения питания имеет один код. В данном объекте управления установлена определенная последовательность включения цепей:

а) скорость из набора V_n;

б) путь из набора S_n;

в) напряжение питания U. Нарушение последовательности не приведет к движению объекта.

2) Описание модели объекта и моделирующей программы

Моделирующая программа в целях упрощения поддерживает набор только из двух скоростей (V_1 и V_2), двух путей (S_1 и S_2) и напряжения питания U.

Необходимо зарезервировать адреса ячеек ОЗУ с помощью переменных RDV, RDS, RDU, FV, FS и FU,
где RD – регистр данных;

F – флаги;

V, S, U – скорость, путь и напряжение.

Основные коды приведены в табл. 3.1.

Таблица 3.1

Управляющие коды моделирующей программы

№	Наименование узлов	Регистр	Моделирующий адрес	Разреш. код	Параметр	Значение в 16 с/с
1	Узел скорости	F RD	FV RDV	1	V1 V2	E0 E8
2	Узел пути	F RD	FS RDS	1	S1 S2	F0 F8
3	Узел напряжения	F RD	FU RDU	1	U	C0

Задание:

1. Разработать основную управляющую программу для управления моделью объекта (по варианту) с использованием моделирующей подпрограммы, используя три параметра объекта.

4. УПРАВЛЕНИЕ ПЕЧАТАЮЩИМ УСТРОЙСТВОМ. КОДИРОВАНИЕ АЛФАВИТНО-ЦИФРОВОЙ ИНФОРМАЦИИ. ПАРАЛЛЕЛЬНЫЕ ПОРТЫ

IBM совместимые компьютеры, имеют единую кодировку символов, т. е. таблицу кодов, в которой каждому изображаемому на экране символу соответствует код от 0 до 255 на основании ASCII-кода (American standart code for information interchange – американский стандартный код для обмена информацией; внедрен в 1963 году). В связи с этим программы, которые выводят на экран сообщения на английском языке, будут работать одинаково и не зависят от того, какая кодировка символов используется в компьютере.

Коды от 0 до 31 и 127 являются управляющими. Эти коды приведены в табл. 4.1.

Таблица 4.1

ASCII-коды управляющие

Коды		Символ	Клавиши	Назначение
10 с/с	16 с/с			
0	00	NUL	@	Машинный нуль
1	01	SON	A	Начало заголовка (start of heading)
2	02	STX	B	Начало текста (start of text)
3	03	ETX	C	Конец текста (end of text)
4	04	EOT	D	Конец передачи (end of transmission)
5	05	EMQ	E	Справка (enquiry)
6	06	ASK	F	Подтверждение (acknowledge)
7	07	BEL	G	Звонок (bell)
8	08	BS	H	Шаг назад (back space)
9	09	HT	I	Горизонтальная табуляция (tab horizontally)
10	0A	LF	J	Перевод строки (line feed)
11	0B	VT	K	Вертикальная табуляция (tab vertically)
12	0C	FF	L	Подача формата (form feed)
13	OD	CR	M	Возврат каретки (carridge return)
14	0E	SQ	N	Внешнее перемещение (shift out)
15	0F	SI	O	Внутреннее перемещение (shift in)
16	10	DLE	P	ESC – последовательность (data line escape)

Окончание табл. 4.1

Коды		Символ	Клавиши	Назначение
10 с/с	16 с/с			
17	11	DC1	Q	Управление 1 (device control 1)
18	12	DC2	R	Управление 2 (device control 2)
19	13	DC3	S	Управление 3 (device control 3)
20	14	DC4	T	Управление 4 (device control 4)
21	15	NAK	U	Отрицательное подтверждение (negative acknowlege)
22	16	SIN	V	Синхронизация (synchronous indl)
23	17	ETB	W	Конец передаваемого текста (end of transmited block)
24	18	CAN	X	Отмена линии (cancel line)
25	19	EM	Y	Переход через середину (end of medium)
26	1A	SUB	Z	Конец текстового файла (substitute)
27	1B	ESC	I	Символ ESC (escape)
28	1C	FS	\	Разделитель файла (file separator)
29	1D	GS	I	Разделитель групп (group separator)
30	1E	RS	^	Разделитель записей (recorol separator)
31	1F	US	-	Разделитель единиц (unit separator)
127	7F	DEL	8	Забой (удаление символа) (delete)

Для некоторых команд используются управляющие ASCII-коды, например: 7 – звонок, 10 – перевод строки, 13 – возврат каретки, 24 – отмена строки, 127 – забой знака.

Для большинства из них формируются так называемые ESC-последовательности, т. е. наборы байтов, первый из которых имеет код ESC (десятичный номер 27). После ESC следует байт с кодом команды, которым может быть любой символ ASCII-кода. Затем идут байты с параметрами команды (если они необходимы).

С помощью ESC-последовательностей можно устанавливать большое число различных шрифтов, задавать графический режим, определять любые собственные знаки, выравнивать текст на странице, устанавливать интервал между строками и символами, производить табуляцию, выбирать различные наборы знаков для расширенной таблицы ASCII-кода и т. д.

Символы с кодами от 32 до 126 приведены в табл. 4.2.

Таблица 4.2

ASCII-коды печатаемые

Код	Символ	Код	Символ	Код	Символ	Код	Символ	Код	Символ	Код	Символ
32	Про	48	0	64	@	80	P	96	'	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	Ї
47	/	63	?	79	O	95	_	111	o		

Управление печатающим устройством, которое относится к периферийным устройствам (ПУ) компьютера, осуществляется так же, как и любым внешним устройством, подключенным к системной шине.

Принтеры, как правило, являются параллельными устройствами и для подключения к системному блоку используют параллельный интерфейс, в соответствии с которым информация передается побайтно.

Каждое параллельное устройство имеет имя LPTn (n = 1, 2...) и свой адаптер, управляемый тремя регистрами ввода-вывода: регистром данных, регистром статуса и регистром управления. Адреса портов этих регистров различны для каждого адаптера. Базовые адреса для каждого адаптера находятся в области данных BIOS (базовая система ввода-вывода, находящаяся в постоянной памяти компьютера). В таблице 4.3 даны адреса портов LPTn.

Таблица 4.3

Адреса портов LPTn

Адрес в 16 с/с	Количество байтов	Содержимое памяти
0:0408	2	Адрес порта LPT 1
0:040A	2	Адрес порта LPT 2
0:040C	2	Адрес порта LPT 3
0:040E	2	Адрес порта LPT 4

Напомним, что адрес занимает 4 байта: 2 байта со старшими адресами определяют сегмент, а 2 байта с младшими адресами – смещение. В таблице 4.3 смещение взято относительно сегмента с адресом 0.

При инициализации базовому адресу присваивается значение 0, когда соответствующий адаптер не установлен. Программа, которая прямо адресуется в параллельный порт, должна искать используемые им адреса.

Каждый посылаемый в принтер байт передается через регистр данных, который имеет 8 разрядов. Программа пользователя должна вычислить номер порта выходных данных в зависимости от выбранного LPTn.

Если выбрать канал LPT1, то тогда, согласно табл. 4.3, необходимо обратиться в область BIOS памяти по адресу 0:0408. Прочитанное определенным образом значение после дополнительного преобразования даст номер порта выходных данных, т. е.:

$$NPORTD = \left[\left\langle 0:0408 \right\rangle \right], \qquad (4.1)$$

где NPORT (D) – номер порта данных;

< > – угловые скобки указывают содержимое ячейки по адресу;

[ ] – квадратные скобки указывают на дополнительное преобразование.

На рис. 4.1 изображены 4 байта, где показаны сегмент и смещение и их адреса соответственно 0 и 0408 для LPT1.

Из рис. 4.1 видно, что в адресе, указанном в табл. 4.3 для LPT1, в качестве смещения приводится адрес самого младшего байта, а именно: 0408. Адрес второго младшего байта автоматически считается рав-

ным 0409. Очевидно, что младший разряд байта с адресом 0409 больше единицы младшего разряда байта с адресом 0408 в $2^8 = 256$ раз.

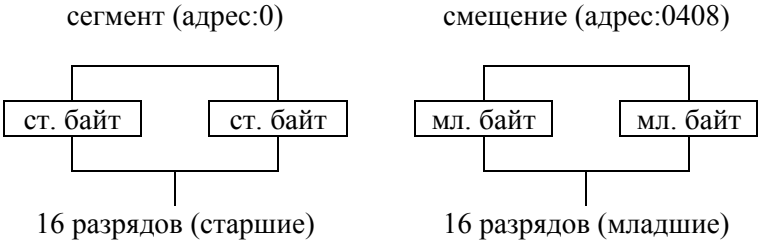


Рис. 4.1. Адреса сегмента и смещения

Таким образом, если содержимое младшего байта с адресом 0408 обозначить через PORTD1, а содержимое байта с адресом 0409 – через PORTD2, то

$$NPORTD = PORTD1 + PORTD2 \cdot 256, \qquad (4.2)$$

причем PORTD1 = < 0408 >; PORTD2 = < 0409 >.

Формула (4.2) показывает алгоритм дополнительного преобразования, который содержится в формуле (4.1).

Таким образом, для определения номера порта, в частности LPT1, необходимо:

– задать текущий адрес сегмента;

– определить содержимое PORTD1;

– определить содержимое PORTD2;

– определить номер порта данных PORTD по формуле (4.2).

Регистр статуса (PC) сообщает различную информацию. Он имеет 8 разрядов. На рис. 4.2 показан формат PC. Разряды с 0 – 2 – не используются.

3-й разряд. Ошибка принтера: «0» – ошибка; «1» – отсутствие ошибки.

4-й разряд. Связь ПЭВМ: «0» – принтер не связан с ПК (off-line); «1» – принтер связан с ПК (on-line).

5-й разряд. Бумага: «0» – бумага вставлена; «1» – нет бумаги.

6-й разряд. Подтверждение приема символа: «0» – принтер подтверждает прием символа; «1» – нормальная установка.

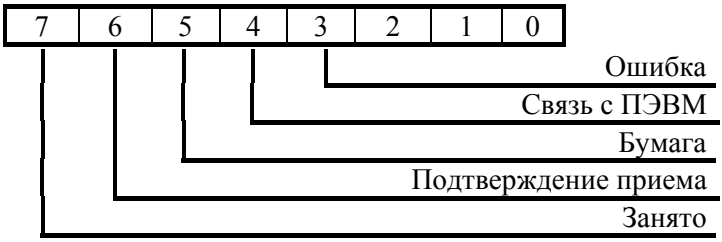


Рис. 4.2. Формат регистра статуса

7-й разряд. Занято: «0» – принтер занят; «1» – принтер свободен.

Неиспользуемые разряды 0–2 имеют единичное состояние. Если принтер выключен, то все разряды, кроме 3-го, установлены в «1», что соответствует коду $F7_{16}$. Если принтер не готов к работе (7-й разряд в «0»), то 3-й разряд имеет «0», что соответствует коду 57_{16}. Если отсутствует бумага, то код 77_{16}.

Двоичный код готовности принтера к работе: $\mathbf{11011111_2 = DF_{16}}$.

Перед обращением к принтеру необходимо проводить анализ на его готовность. Это можно делать с помощью оператора WAIT (ждать пока). Этот оператор приостанавливает процесс выполнения программы до тех пор, пока в порту с адресом A не появится кодовая комбинация, удовлетворяющая $n1$, которая задает условие прекращения задержки.

Номер порта регистра статуса вычисляется по формуле:

$$NPORTS = NPORTD + 1, \tag{4.3}$$

т. е. номер порта регистра статуса (NPORTS) отличается от номера порта регистра данных на 1 в сторону увеличения.

Тогда оператор WAIT можно записать следующим образом:

WAITNPORTS, &HDF. (4.4)

При выполнении этого оператора может быть начата передача данных в регистр данных. Однако до появления этого оператора в программе пользователя необходимо выполнить определенные операции с принтером, то есть инициализировать принтер через регистр управления.

Регистр управления (РУ) устанавливает адаптер в исходное состояние и координирует вывод данных. Регистр имеет 8 разрядов. Его формат показан на рис. 4.3.

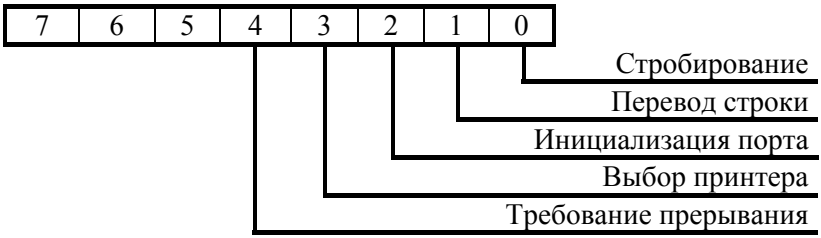


Рис. 4.3. Формат регистра управления

0-разряд. Стробирование: «0» – нормальная установка; «1» – кратковременное единичное значение воспринимается как стробирующий сигнал для вывода байта.

1-й разряд. Перевод строки: «0» – нормальная установка; «1» – автоматический перевод строки после возврата каретки.

2-й разряд. Инициализация порта принтера: «0» – инициализация; «1» – нормальная установка.

3-й разряд. Выбор принтера: «0» – отмена выбора принтера; «1» – нормальная установка.

4-й разряд. Требование передачи: «0» – прерывание запрещено; «1» – прерывание разрешено.

5-й – 7-й разряды не используются.

Программа пользователя должна инициализировать порт принтера перед первым его использованием, а в случае появления ошибки необходимо проводить повторную инициализацию.

Для инициализации необходимо в регистре управления сбросить в «0» 2-й разряд (инициализация порта) на 0,05 с. В этот момент в единице должен быть только 3-й разряд (выбор принтера). Это равносильно записи в РУ следующего кода:

$$KI1 = 1000_2 = 8_1, \tag{4.5}$$

где $KI1$ – условно обозначено первое значение кода для инициализации.

После задержки, которая должна быть не менее 1–2 сек, 2-й разряд (инициализация) устанавливается в "1" (нормальная установка), при этом 3-й разряд остается в "1". Это соответствует записи в РУ следующего кода:

$$KI2 = 11002 = C16, \tag{4.6}$$

где *KI*2 – условно обозначено второе значение кода при инициализации.

Номер порта регистра управления вычисляется по формуле:

$$NPORTU = NPORTD + 2, \tag{4.7}$$

где NPORTU – номер порта регистра управления, а NPORTD вычислен в (4.2).

После инициализации в программе следует использовать оператор присваивания номера символа, который желает напечатать пользователь.

Например:

$$SYMBOL = < \text{код} >, \tag{4.8}$$

где код выбирается по табл. 4.1–4.3; код удобно задавать в 10 с/с.

Например: SYMBOL = 33_{10} – символ «восклицательный знак».

Прежде, чем записать значение SYMBOL в порт, необходимо проверить состояние регистра статуса.

Если получено разрешение на выполнение последующих операторов программы, то следующим оператором должен быть оператор записи в порт данных заданного символа.

Вслед за посылкой байта символа в порт выходных данных необходимо организовать стробирование РУ (рис. 4.3). Кратковременная установка 0-го разряда РУ в "1" осуществляется аналогично операции инициализации: в порт РУ послать друг за другом два кода:

$$\begin{gathered} KCTP1 = 1100 + 1 = 1101 = D16; \\ KCTP2 = 1100 = C16. \end{gathered} \tag{4.9}$$

После посылки байта и включения стробирующего сигнала программа должна ждать сигнал готовности принтера. Подтверждающий сигнал выражается не только в изменении 7-го разряда (занято); дополнительно на короткое время будет сброшен в "0" 6-й разряд регистра статуса, что свидетельствует о подтверждении приема сигнала принтером.

После этого может быть послан следующий байт символа. Если символы выводятся подряд, то следующим оператором может быть подготовка очередного символа кодировочной таблицы (см. табл. 4.2):

$$SYMBOL = SYMBOL + 1 . \qquad (4.10)$$

Пример алгоритма управления печатающим устройством для вывода символов в цикле:

1 – Определение адреса сегмента.

2 – Вычисление номеров портов регистра данных, регистра статуса и регистра управления.

3 – Инициализация порта управления.

4 – Выбор символа для печати.

5 – Ожидание готовности.

6 – Запись символа в регистр выходных данных.

7 – Организация стробирующего сигнала.

8 – Ожидание готовности.

9 – Подготовка следующего символа.

10 – Формирование условия окончания вывода символов.

После выполнения условия шага 10 и сформированной управляющей последовательности управление передается к шагу 5.

Задание: Составить программу для вывода на принтер символов согласно табл. 4.2. (коды 32–126, коды 128–255).

5. БИТОВЫЕ ОПЕРАТОРЫ. СПОСОБЫ ОПИСАНИЯ ЛОГИЧЕСКИХ ФУНКЦИЙ (БУЛЕВА АЛГЕБРА)

Булева алгебра или алгебра логики – это раздел математики, изучающий высказывания, рассматриваемые со стороны их логических значений (истинности или ложности) и логических операций над ними. Алгебра логики позволяет сформулировать и закодировать любые утверждения, истинность или ложность которых нужно доказать и использовать их как обычные числа в математике.

Значениям переменной в булевой алгебре соответствуют состояниям элементов микросхем компьютера или любого другого электронного устройства: сигнал присутствует (логическая "1") или сигнал отсутствует (логический "0").

На логических элементах, реализующих булевы функции, строятся логические схемы электронных устройств.

Применяются следующие способы описания логических функций: словесный, табличный, числовой, аналитический, координатный, графический. Рассмотрим некоторые из них.

Табличное описание логических функций:

x_1	x_2	f
0	0	1
0	1	0
1	0	1
1	1	0

Данные таблицы имеют название таблиц истинности.

Аналитическое описание логических функций:

$$f = x_1 \wedge x_2 \vee x_3 \wedge \neg x_1 \vee \neg x_3. \qquad (5.1)$$

Любую булеву функцию с произвольным количеством аргументов можно построить через подстановку элементарных функции вместо аргументов (суперпозицию). Набор простейших функций, с помощью которого можно выразить любые другие, сколь угодно сложные логические функции, называется функционально полным набором, или логическим базисом (НЕ, И, ИЛИ).

Инверсия – логическое отрицание (НЕ)

$$f(x) = \neg x, \tag{5.2}$$

x	$\neg x$
0	1
1	0

Конъюнкция – логическое умножение (И)

$$f(x_1, x_2) = x_1 \wedge x_2, \tag{5.3}$$

x_1	x_2	f
0	0	0
0	1	0
1	0	0
1	1	1

Дизъюнкция – логическое сложение, (ИЛИ)

$$f(x_1, x_2) = x_1 \vee x_2, \tag{5.4}$$

x_1	x_2	f
0	0	0
0	1	1
1	0	1
1	1	1

Приоритеты операций:

1. Инверсия
2. Конъюнкция
3. Дизъюнкция

В булевом базисе обычно строятся логические схемы, которые реализуют сколь угодно сложные логические функции.

Графический способ описания подразумевает построение логических схем. Логические схемы создаются для реализации в цифровых устройствах булевых функций.

Логические схемы реализуются на логических элементах: "НЕ", "И", "ИЛИ", "И-НЕ", "ИЛИ-НЕ", "Исключающее ИЛИ" и "Эквивалентность".

Для обозначения логических элементов используется несколько стандартов. Наиболее распространенными являются ANSI – American national standards institute (Американский национальный институт стандартов), DIN – Deutsches Institut für Normung (Немецкий институт по стандартизации), IEC – International Electrotechnical Commission (Международная электротехническая комиссия) и российский ГОСТ.

Согласно ГОСТ и IEC блоки логических элементов выглядят следующим образом (рис. 5.1).

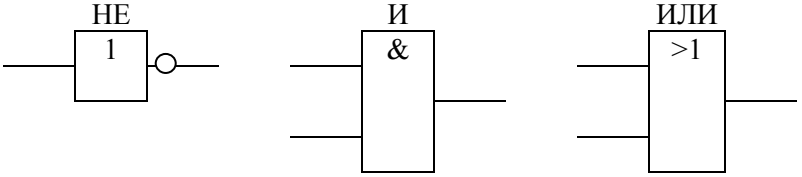


Рис. 5.1. Блоки логических элементов булева базиса

Для программной реализации алгебры логики в C++ существуют битовые операторы.

Битовые операции – это тестирование, установка или сдвиг битов в байте или слове. Битовые операторы используются только со стандартными типами – char, bool и int и не могут использоваться с float, double, long double, void.

В табл. 5.1 приведены битовые операторы языка C++.

Таблица 5.1

Битовые операторы языка C++

Оператор	Логическая функция
&	Конъюнкция И
|	Дизюнкция ИЛИ
^	Исключающее ИЛИ
!	Отрицание НЕ

Задание:

1. Составить таблицу истинности и построить логическую схему для логической функции по варианту (табл. 5.2).

2. Написать программу, реализующую решение данной функции.

Пример программы:

```
#include <iostream.h>
int main()
{
        bool x1[4]={true,true,false,false};
        bool x2[4]={true,false,true,false};
        bool y[4];
        for(int i=0;i<4;i++)
        {
        y[i]=x1[i]&x2[i];
        y[i]=y[i]^x1[i];
        y[i]=!y[i];
        y[i]=y[i]|x1[i];
                cout<<y[i]<<"\t";
        }
        return 0;
}
```

Варианты заданий приведены в табл. 5.2.

Таблица 5.2

Варианты заданий

1	$f = (\neg(x_1 \wedge x_2) \vee ((x_1 \wedge x_2) \wedge \neg x_2)) \vee \neg(x_1 \vee x_2)$
2	$f = (\neg(x_1 \vee x_2) \wedge x_2) \wedge (\neg((x_1 \wedge x_2) \vee \neg x_1))$
3	$f = (\neg(\neg x_1 \wedge x_2) \wedge \neg x_2) \wedge ((x_1 \vee \neg x_2) \vee \neg x_1)$
4	$f = (\neg((x_1 \wedge \neg x_2) \vee (\neg x_1 \vee x_2))) \vee (\neg(\neg x_1 \vee x_2) \vee x_2)$
5	$f = (\neg((x_1 \wedge x_2) \wedge (x_1 \vee x_2))) \wedge (\neg(x_1 \vee x_2) \vee x_1)$
6	$f = ((\neg x_1 \wedge x_2) \wedge \neg x_1) \vee (\neg(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2))$
7	$f = (\neg(\neg(x_1 \vee x_2) \wedge x_2)) \wedge (x_1 \vee \neg(x_1 \wedge x_2))$
8	$f = (\neg(\neg x_1 \wedge \neg x_2) \vee \neg(x_1 \wedge x_2)) \wedge ((\neg x_1 \vee x_2))$
9	$f = (((\neg x_1 \vee x_2) \wedge x_2) \vee \neg x_1) \wedge (\neg(x_1 \vee \neg x_2)) \wedge \neg x_2)$
10	$f = (\neg(\neg(\neg(x_1 \wedge x_2)) \vee (x_1 \wedge x_2))) \wedge (\neg(\neg(\neg(x_1 \vee x_2) \wedge (x_1 \vee x_2)))$
11	$f = (\neg(x_1 \wedge \neg x_2) \vee x_2) \wedge x_1) \vee (\neg((\neg x_1 \vee x_2) \vee \neg x_2))$
12	$f = ((\neg(\neg x_1 \wedge \neg x_2) \wedge x_2) \vee (\neg(x_1 \vee x_2))) \wedge (\neg(\neg x_1 \vee x_2))$
13	$f = (\neg(\neg((x_1 \wedge x_2) \wedge x_2) \wedge x_2)) \wedge (x_1 \wedge \neg(x_1 \vee x_2))$

6. УПРАВЛЕНИЕ КЛАВИАТУРОЙ МИКРОЭВМ СИСТЕМЫ INTEL

Клавиатура содержит микропроцессор Intel, который воспринимает каждое нажатие на клавишу и выдает скан-код в порт А микросхемы интерфейса с периферией, расположенной на системной плате. Скан-код – это однобайтовое число, младшие 7 бит которого представляют идентификационный номер, присвоенный каждой клавише. На всех ПК, выпускавшихся до АТ, старший бит кода говорит о том, была ли клавиша нажата (бит = 1, код нажатия) или освобождена (бит = 0, код освобождения). Например, 7-битовый скан-код клавиши В – 48, или 1z10000 в 2 с/с. Когда эту клавишу нажимают, в порт А посылается код 10110000, а когда отпускают – код 00110000. Таким образом, каждое нажатие на клавишу дважды регистрируется. И всякий раз микросхема выдает подтверждение процессору клавиатуры. Начиная с АТ все происходит по-другому, в обоих случаях посылается один и тот же скан-код, однако если клавиша отпускается, то предварительно следует код $F0_{16}$.

Когда скан-код выдается в порт А, вызывается прерывание клавиатуры (INT 9). Процессор прекращает свою работу и выполняет процедуру, анализирующую скан-код. При поступлении кода от клавиши сдвига или переключателя изменение статуса записывается в память.

Во всех остальных случаях скан-код трансформируется в код символа при условии, что он подается при нажатии клавиши (в противном случае скан-код отбрасывается). Процедура сначала определяет установку клавиш сдвига и переключателей, чтобы правильно получить вводимый код (например, "а" или "А"). Затем введенный код помещается в буфер клавиатуры, который является областью памяти, способной запомнить до 15 вводимых символов, пока программа слишком занята, чтобы обработать их. На рис. 6.1 показан путь, который проходит нажатие на клавишу перед тем, как попасть в программу.

Имеются два типа кодов символов: коды ASCII и расширенные коды. Коды ASCII – это байтовые числа, соответствующие расширенному набору кодов ASCII для IBM PC.

Существуют несколько комбинаций клавиш для выполнения специальных функций, они не генерируют скан-коды. Эти комбинации включают "Ctrl-Break", "Ctrl-Alt-Del", "PrtSc" и другие. Эти исключения приводят к заранее предопределенным результатам.

Все остальные нажатия клавиш должны интерпретироваться программой, и если они имеют специальное назначение, например, сдвинуть курсор влево, то программа должна содержать код для достижения этого эффекта.

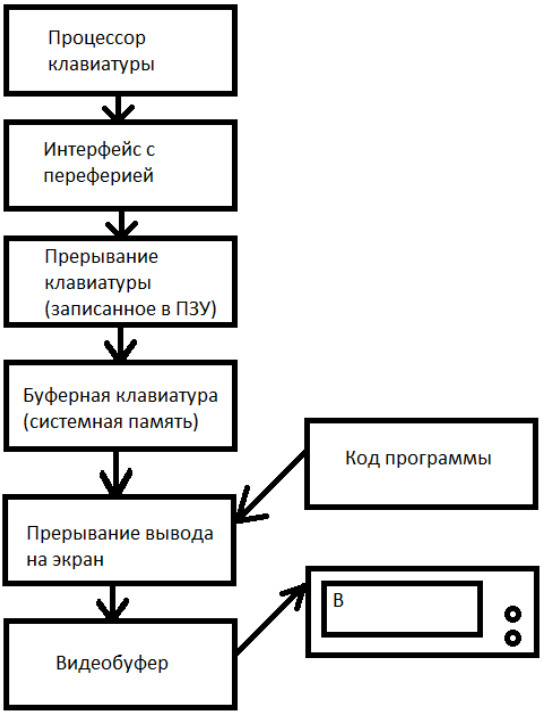


Рис. 6.1. Процесс отображения символа на экране монитора после нажатия соответствующей клавиши клавиатуры

Буфер клавиатуры может накапливать до 15 нажатий на клавишу независимо от того, являются ли они однобайтовыми кодами ASCII или двухбайтовыми расширенными кодами. Таким образом, буфер должен отвести два байта памяти для каждого нажатия на клавишу. Для однобайтовых кодов первый байт содержит код ASCII, а второй – скан-код клавиши. Для расширенных кодов первый байт совпадает со скан-кодом клавиши, но не всегда, поскольку некоторые клавиши можно комбинировать с клавишами сдвига для генерации различных кодов.

Буфер устроен как циклическая очередь, которую называют также буфером FIFO (первый вошел – первый ушел). Он занимает непрерывную область адресов памяти. Однако определенной ячейки памяти, которая хранит "начало строки", в буфере нет. Вместо этого два указателя хранят позиции головы и хвоста строки символов, находящихся в буфере в текущий момент. Новые нажатия клавиш сохраняются в позициях, следующих за хвостом (в более старших адресах памяти), и, соответственно, обновляется указатель хвоста буфера. После того, как израсходовано все буферное пространство, новые символы продолжают вставляться, начиная с самого начала буферной области; поэтому возможны ситуации, когда голова строки в буфере имеет больший адрес, чем хвост. Когда буфер заполнен, новые вводимые символы игнорируются, при этом прерывание клавиатуры выдает гудок через динамик.

Для очистки буфера надо просто установить значение ячейки 0040:001A равным значению ячейки 0040:001C.

Байты статуса расположены в ячейках памяти 0040:0017, 0040:0018 и содержат биты, отражающие статус клавиш сдвига и других клавиш переключателей следующим образом:

Таблица 6.1

Байты статуса клавиатуры

	Бит	Клавиша	Значение, когда бит равен 1
0040:0017	7	Insert	режим вставки включен
	6	CapsLock	режим CapsLock включен
	5	NumLock	режим NumLock включен
	4	ScrollLock	режим ScrollLock включен
	3	Alt	клавиша нажата
	2	Ctrl	клавиша нажата
	1	левый Shift	клавиша нажата
	0	правый Shift	клавиша нажата
0040:0018	7	Insert	клавиша нажата
	6	CapsLock	клавиша нажата
	5	NumLock	клавиша нажата
	4	ScrollLock	клавиша нажата
	3	Ctrl-NumLock	режим Ctrl-NumLock включен
остальные биты не используются			

Прерывание клавиатуры немедленно обновляет эти биты статуса, как только будет нажата одна из клавиш переключателей, даже если не было считано ни одного символа из буфера клавиатуры. Это верно и для клавиши "Ins", которая единственная из этих 8 клавиш помещает код в буфер (установка статуса "Ins" меняется, даже если в буфере нет места для символа). Бит 3 по адресу 0040:0018 устанавливается в 1, когда действует режим задержки "Ctrl-NumLock", поскольку в этом состоянии программа приостановлена, этот бит несуществен.

Прерывание клавиатуры проверяет состояние статусных битов перед тем, как интерпретировать нажатые клавиши, поэтому, когда программа меняет один из этих битов, эффект тот же, что и при физическом нажатии соответствующей клавиши. Например, программа пользователя может нуждаться в чтении статуса этих клавиш, например, для того, чтобы вывести текущий статус на экран.

Основное отличие клавиатуры от некоторых других устройств ввода/вывода с точки зрения управления состоит в том, что она, как правило, жизненно необходима для общения с компьютером. Пользователь может либо полностью взять на себя управление клавиатурой, либо только обрабатывать данные, подаваемые управляющей программой. Это результат того, что клавиатура имеет собственное прерывание.

Без сигнала прерывания невозможно предсказать, что именно находится в порту.

Некоторые из режимов имеют световые индикаторы на клавиатуре: CapsLock, NumLock, ScrollLock, поэтому о смене режима интерпретации нажимаемых клавиш можно судить визуально - по световым индикаторам.

В задачу данной лабораторной работы входит программная проверка установленных режимов клавиатуры.

Два байта по адресам 0040:0017 и 0040:0018 являются байтами статуса. Поэтому, если, например, необходимо включить режим заглавных букв (CapsLock), следует установить бит 6 байта по адресу 0040:0017. Менять необходимо только те биты, которые нужны, а оставшиеся необходимо оставить в исходном виде. Для обеспечения этого используются логические операции: AND, OR. Для установки 6 бита указанного байта статуса, необходимо сначала прочитать его, а затем с помощью операции ИЛИ (OR) установить 6-ой бит, записав

полученный байт обратно в память. Символами вопроса отмечены те биты, содержание которых неизвестно и не должно поменяться:

?	0	?	?	?	?	?	?	– содержание байта статуса
OR (ИЛИ)								
0	1	0	0	0	0	0	0	– операнд, необходимый для установки 6-ого бита

$$NSTAT = STAT\ OR\ 64. \qquad (6.1)$$

Если же, например, надо выключить режим CapsLock, то следует сбросить 6-ой бит. После выполнения логического И (AND) 6-ой бит обязательно сбросится, тогда как остальные останутся неизменными:

?	1	?	?	?	?	?	?	– содержание байта статуса
AND (И)								
1	0	1	1	1	1	1	1	– операнд, необходимый для установки 6-ого бита

В переменной STAT находится прочитанное из памяти значение байта статуса:

$$...NSTAT = STAT\ AND\ 191 \qquad (6.2)$$

191 – это число, соответствующее в 10 с/с числу 10111111_2. После выполнения логической операции байт необходимо записать в память по тому же адресу, по которому он был считан.

Если необходимо какие-либо биты сбросить, а какие-то установить, потребуется произвести подряд несколько логических операций с помощью AND и OR.

Задание:

Написать программу, которая проверяет состояние работы клавиатуры в соответствии с заданным вариантом. Структура байтов статуса приведена в табл. 5.1.

Варианты и задания к ним приведены в табл. 6.2. Листинг программы включить в отчет.

Таблица 6.2

Варианты заданий

Вариант	Режим работы клавиатуры
1	CapsLock выключен, ScrollLock выключен, NumLock выключен
2	CapsLock выключен, ScrollLock включен, NumLock выключен
3	CapsLock включен, ScrollLock выключен, NumLock выключен
4	CapsLock включен, ScrollLock включен, NumLock выключен
5	CapsLock включен, ScrollLock включен, NumLock включен
6	CapsLock включен, ScrollLock выключен, NumLock включен
7	CapsLock выключен, ScrollLock включен, NumLock включен
8	CapsLock выключен, ScrollLock выключен, NumLock включен

7. СИСТЕМНЫЙ РАСЧЕТ АНАЛОГОВО-ЦИФРОВОГО ПРЕОБРАЗОВАТЕЛЯ

Аналогово-цифровой преобразователь (АЦП) является неотъемлемой и важнейшей частью любой микропроцессорной контрольно-измерительной, информационной и управляющей системы.

АЦП служит для преобразования входной аналоговой величины от объекта контроля и/или управления в соответствующий ей цифровой эквивалент-код, который является его входным сигналом.

В основе классификации АЦП лежат два признака. Главным признаком классификации является алгоритм его работы или метод преобразования. Вторым признаком является род аналоговой величины.

Наибольшее распространение на практике нашли три классических метода преобразования: метод последовательного счета, метод поразрядного кодирования и метод считывания.

Метод последовательного счета – входная аналоговая величина уравновешивается суммой одинаковых минимальных эталонов, называемых квантами.

Метод поразрядного кодирования – входная величина последовательно сравнивается с суммой эталонов, имеющих 2^j элементов, где

$j = n - 1, n - 2, ...2, 1, 0$ (где n – количество разрядов в двоичном коде).

Метод считывания – здесь используется набор из $2^n - 1$ эталонов:

младший эталон равен одному кванту, следующий – двум квантам и старший – $(2^k) - 1$ квантам.

Наибольшее распространение на практике нашли следующие виды аналоговых величин: напряжение и ток, угловое и линейное перемещение, временной интервал, фаза и частота переменного тока.

АЦП выполняются в настоящее время в виде платы – модуля на основе ИС, или в одном кристалле (БИС), или встроенные в кристалл БИС микроконтроллера.

Преобразователи перемещений в коды занимают особое место по отношению к АЦП, использующих только физические явления, характерные для электронных систем, т. к., во-первых, применяют различные физические явления, приводящие к наличию или отсутствию контакта, поля, светового луча и т. д. Во-вторых, из-за сложности конструкции их характеристики во многом определяются совершенством технологии. В них, как правило, используются два

метода преобразования: последовательного счета и считывания. Вопросами создания новых видов преобразователей угла в код занимаются специалисты по точной механике.

В АЦП используются следующие виды кодов: двоичные, десятичные коды, отраженные (рефлексные), в частности, наиболее распространённый код Грея, и корректирующие коды.

В непрерывном входном сигнале (ВС), поступающем из входного канала (ВК), для любого момента времени, во-первых, имеется отличная от нуля информация, т. е. он является непрерывным во времени. Во-вторых, его величина может иметь, в принципе, бесконечно большое число значений.

Дискретный сигнал фиксируется на небольшом конечном интервале времени, а его амплитуда может принимать определенное конечное число значений. Поэтому переход от непрерывного ВС к дискретному всегда связан с его округлением, что приводит к ошибкам двух видов:

– от квантования по уровню;

– от квантования по времени.

Квантование по уровню (дискретизация по уровню) – это замена точного значения величины сигнала приближенным дискретным значением, при котором два ближайших разрешенных дискретных значения различаются на элементарную величину (квант).

Квантование по времени (дискретизация по времени) – это замена непрерывного во времени сигнала дискретными сигналами, поступающими (например, на АЦП) в определенные моменты времени прерывисто.

Для АЦП характерны три основные погрешности: погрешность квантования по времени, погрешность квантования по уровню и инструментальная погрешность.

Инструментальная погрешность появляется из-за шумов и помех, как во входном сигнале, так и в узлах АЦП из-за технологических отклонений, возникающих при изготовлении и эксплуатации АЦП. Точность изготовления цифровых элементов практически не влияет на точность преобразования. Механические перемещения (угловые и линейные) обычно задаются на порядок точнее по сравнению с электрическими, чтобы свести практически на нет их влияние на общую погрешность АЦП. Таким образом, основное влияние оказывает наличие случайной погрешности во входном сигнале, т. к.:

$$U_{\text{вх}} = U_n + E, \tag{7.1}$$

где $U_{\text{вх}}$ – входной сигнал АЦП;

U_n – полезный сигнал;

E – случайный сигнал (помеха).

Отметим, что в промышленных условиях всегда в той или иной мере на полезный сигнал накладывается помеха. Для повышения точности, надежности и достоверности получаемой информации по входным каналам существует много способов:

– использование унифицированных сигналов;

– скрутка двухпроводных линий связи;

– гальваническая развязка;

– аналоговая фильтрация на входе АЦП;

– цифровая фильтрация выходных сигналов и другие.

Если имеется случайная погрешность, то однозначное соответствие между входным сигналом и кодом теряется: ВС по-разному накладывается на сетку разрешенных уровней квантования. Поэтому, зная выходной код, можно говорить только о вероятности того или иного значения входной величины в пределах данного кванта.

Шаг квантования по уровню представляет собой разрешающую способность АЦП, которая вычисляется по формуле:

$$D = \frac{U_{\max} - U_{\min}}{N}, \tag{7.2}$$

где $U_{\max}$ и $U_{\min}$ – максимальное и минимальное значение преобразуемой величины;

N – число разрешенных уровней квантования.

Обозначим:

$$U_{\max} - U_{\min} = P, \tag{7.3}$$

где P – диапазон изменения входного сигнала.

Тогда

$$D = P/N, \tag{7.4}$$

Число N зависит от разрядности АЦП:

$$N = 2^R - 1, \tag{7.5}$$

где R – количество разрядов АЦП.

Из (7.5) следует:

$$R = \log_2\left(\frac{P}{\delta} + 1\right), \tag{7.6}$$

Формула (7.6) позволяет определить разрядность АЦП. Однако надо знать N, которое может быть вычислено по формуле (7.4), если известны значения P и D. Если D неизвестно, то оно рассчитывается через среднеквадратичную погрешность квантования по уровню $G2$.

Абсолютная ошибка квантования по уровню изменяется в пределах от –0.5*D до 0.5*D и с равной вероятностью может принимать любое значение в этом диапазоне.

Следовательно, для ошибки в точке отсчета можно принять равномерный (прямоугольный) закон распределения, для которого дисперсия ошибки квантования по уровню:

$$\delta = \left(\frac{D}{2}\right)^{\frac{2}{3}}, \tag{7.7}$$

Отсюда СКП квантования по уровню равна:

$$\sigma_u = \frac{D}{2\sqrt{3}}, \tag{7.8}$$

Если известна σ_u, то

$$D = 2\sqrt{3} \cdot \sigma_u, \tag{7.9}$$

Округление D следует производить в меньшую сторону.

СКП квантования по времени G3 зависит от двух факторов:

– от инерционности АЦП, т. е. от переходных процессов в АЦП;

– от скорости изменения ВС в процессе преобразования.

Обычно ошибка, вызванная первым фактором, при правильно выбранных временных соотношениях в АЦП намного меньше ошибок, вызванных вторым фактором. Поэтому характер изменения ВС, его частотные свойства являются основным фактором при выборе частоты квантования по времени F.

Может быть несколько подходов при расчете F.

I ПОДХОД: использование теоремы В. А. Котельникова, согласно которой период квантования по времени равен:

$$T = \frac{1}{f} = \frac{1}{2f_{\text{гр}}}, \tag{7.10}$$

где $f_{\text{гр}}$ – граничная частота спектра непрерывного сигнала.

Эта теорема эффективно используется при построении систем передачи информации. Ее применение для выбора необходимой частоты квантования сигнала с неограниченным спектром на конечном временном интервале приводит к очень высоким требованиям к частоте f.

II ПОДХОД: замена преобразуемой величины некоторой аппроксимирующей зависимостью. В этом случае погрешность аппроксимации определяется остаточным членом интерполяционной формулы.

При использовании кусочно-линейной аппроксимации

$$T = \frac{1}{f} = \sqrt{\frac{8\delta}{U''(t)}}, \tag{7.11}$$

где δ – допустимое значение погрешности аппроксимации;

$U''(t)$ – вторая производная по времени от входного сигнала.

Недостаток этого подхода – сложность определения $U''(t)$ и завышенные значения погрешности интерполяции.

III ПОДХОД: геометрический.

Можно считать, что за время преобразования входная величина не должна изменяться больше, чем на квант:

$$\frac{\Delta U}{\Delta t} \cdot T \le D,$$

откуда

$$T \le D/\Delta U, \tag{7.12}$$

где ΔU – максимальная скорость изменения входной величины.

Этот подход может использоваться при грубых (прикидочных) расчетах.

Недостаток – не учитывает статистических характеристик сигнала и требует значения ΔU.

IV ПОДХОД: учитывает то, что, как правило, известно об объекте: диапазон изменения входного сигнала P и верхнюю частоту в спектре сигнала $f_{гр}$ или $\omega = 2\pi f_{max}$. При этом форма спектра неизвестна.

Обычно в информационно-измерительных и управляющих системах приходится иметь дело с сигналами, которые можно рассматривать как стационарные случайные процессы с двумя видами распределения: равномерным (прямоугольным) и нормальным.

Для случайного сигнала с равномерным законом распределения в диапазоне P.

$$\sigma_t = \frac{\omega TP}{6}, \tag{7.13}$$

Из (7.13), если известна σ_t

$$T = \frac{6\sigma_t}{\omega P}, \tag{7.14}$$

Для сигнала со спектром, имеющим нормальное распределение:

$$\sigma_t = \frac{\omega TP}{10,4}, \tag{7.15}$$

Из (7.15) следует:

$$T = \frac{10,4\sigma_t}{\omega P}, \tag{7.16}$$

V ПОДХОД: при использовании не мгновенных, а усредненных значений входного сигнала можно использовать формулу, не учитывающую диапазон изменения сигнала

$$\sigma_t = \frac{T^2\omega^2}{24}, \tag{7.17}$$

Эта формула предъявляет менее жесткие требования к частоте квантования:

$$T = \frac{\sqrt{24\sigma_t}}{\omega}, \tag{7.18}$$

Использование АЦП в многоканальном режиме ужесточает требования к частоте квантования в K раз, где K – количество входных каналов, обслуживаемых одним АЦП. В этом случае период квантования T уменьшается, а частота квантования увеличивается в K раз. Причем расчет T необходимо вести по входному каналу с самым быстроизменяющимся сигналом.

Формулы (7.14), (7.16) и (7.18) принимают вид:

$$T = \frac{6\sigma_t}{\omega PK}, \tag{7.19}$$

$$T = \frac{10,4\sigma_t}{\omega PK}, \tag{7.20}$$

$$T = \frac{\sqrt{24\sigma_t}}{\omega K}, \tag{7.21}$$

Суть методики предварительного расчета АЦП состоит:

а) в вычислении среднеквадратичных погрешностей всех составляющих, оказывающих значительное влияние на среднеквадратичную погрешность МСУ ($\sigma_{АЦП}$, σ_U, σ_t) при задании ограничений на каждую из составляющих по отношению к среднеквадратичной погрешности аналогового сигнала канала;

б) в вычислении параметров АЦП по допустимым расчетным составляющим среднеквадратичных погрешностей преобразования с последующим выбором серийного АЦП.

Предварительный расчет учитывает требования к АЦП со стороны разработчика. В этом расчете должна быть произведена оценка или распределение погрешностей по основным составляющим АЦП и самого АЦП.

АЦП не должен вносить значительной погрешности в процесс измерения. Для этого СКП АЦП должна составлять только часть СКП входного канала:

$$\sigma_{АЦП} = k_1 \cdot \sigma_к, \tag{7.22}$$

где k_1 – коэффициент приведения $\sigma_{АЦП}$ к $\sigma_к$.

Это необходимо, потому что полная СКП преобразования входного сигнала рассчитывается по формуле:

$$\sigma_{МСУ} = \sqrt{\sigma_к^2 + \sigma_{АЦП}^2}, \tag{7.23}$$

Подставив (7.22) в (7.23), получим:

$$\sigma_{МСУ} = \sigma_к \sqrt{1 + k_1^2}, \tag{7.24}$$

Из (7.24) следует, что k_1 может изменяться в диапазоне $0 < k_1 < 1$.

Однако только при $k_1 < 0{,}5$ ее вес в общей погрешности будет почти на порядок меньше, чем $\sigma_к$, и не будет вносить существенной погрешности.

СКП квантования по уровню должна составлять только часть СКП АЦП из тех же соображений, которые рассмотрены выше

$$\sigma_U = k_2 \cdot \sigma_{АЦП}, \tag{7.25}$$

где k_2 – коэффициент приведения σ_U к $\sigma_{АЦП}$.

Ошибку σ_t можно уменьшить в большинстве случаев, повысив частоту квантования. В этом случае основной вес будет иметь σ_U. Расчет составляется из

$$\sigma_{АЦП} = \sqrt{\sigma_U^2 + \sigma_t^2}, \tag{7.26}$$

После подстановки в нее *G*2 в соответствие с (7.25):

$$\sigma_t = \sigma_{АЦП} \sqrt{1 - k_2^2}, \tag{7.27}$$

Порядок предварительного расчета АЦП

1. Обычно известен диапазон изменения входного сигнала P и предельная частота ω. Если частотные свойства ВС неизвестны, то необходимо произвести оценку исходя из литературных источников или априорных данных.

2. Исходя из $\sigma_к$ и требований ТЗ на систему, задать коэффициенты k_1 и k_2.

3. Рассчитать σ_U, σ_t, $\sigma_{АЦП}$ и $\sigma_{МСУ}$ исходя из формул (7.25), (7.27), (7.22) и (7.24) соответственно.

4. Рассчитать период T и частоту квантования f. Для этого используются формулы, для многоканального режима АЦП, как более общие: (7.19)–(7.24). При этом, если АЦП работает в одноканальном режиме, то $K = 1$.

Если не известно, будет ли производиться усреднение значений ВС, то необходимо использовать (7.19) или (7.20) для расчета T.

Если характер распределения ВС (случайный или нормальный) неизвестен, то используется (7.19), которая предъявляет более высокие требования к частоте F:

$$f_{АЦП} = \frac{1}{T}, \tag{7.28}$$

5. Расчет разрядности:

– D по формуле (7.9);

– N_1 исходя из формулы (7.4):

$$N_1 = \frac{P}{D}, \tag{7.29}$$

– R_1, используя формулу (7.6).

6. Значение R_1 необходимо округлить в большую сторону.

7. На основании полученных данных $\sigma_{АЦП}$, $f_{АЦП}$ (или T) и R осуществить выбор АЦП. Если серийного АЦП нет, то надо или снизить требования к АЦП или, в случае необходимости, сформулировать ТЗ на разработку нового АЦП.

Задание:

Выполнить выбор АЦП по входным параметрам для трех законов распределения СВ погрешности, варьируя значения:

k_1, k_2 = 0.5, СКП входного канала [0.01–0.1]

а) Диапазон изменения ВС P = const, предельная частота:

$$f = \begin{cases} 1-100\ \text{Гц} \\ 1-100\ \text{кГц} \\ 1-100\ \text{Мгц} \end{cases}$$

Законы распределения: равномерный, нормальный, нормальный со сгаживанием. Количество входных каналов =1 или N.

б) Предельная частота f = const, Диапазон изменения ВС:

$$P = \begin{cases} 1-10 \\ 10-100 \\ 100-10000 \end{cases}$$

Законы распределения: равномерный, нормальный, нормальный со сгаживанием. Количество входных каналов =1 или N.

Проанализировать результаты расчета.

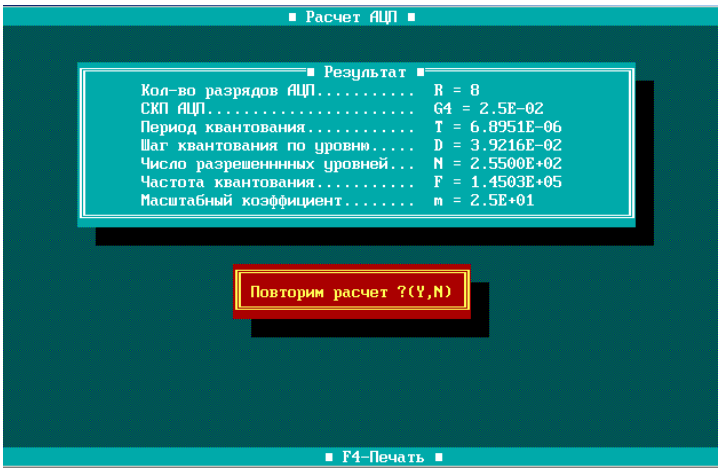


Рис. 7.1. Результаты расчета параметров АЦП

8. ОСНОВЫ РАБОТЫ В VISUAL C++. ВВОД-ВЫВОД. ЛИНЕЙНЫЕ АЛГОРИТМЫ

1. Запустить Visual C++;

2. В меню File выбрать пункт New. На вкладке Workspaces выбрать Blank Workspace (пустое рабочее пространство);

3. Создать проект. File – New – Project. Назначить тип проекта Win32 Console Application. Empty. Добавить проект к созданному рабочему пространству;

4. Добавить к проекту исходный файл программ: File – New – Files. Тип файла – C++ Source File;

5. Ввести текст программы, откомпилировать ее и выполнить:

– Build – Compile или CTRL + F7 () – компилятор проверяет программу на наличие ошибок;

– Build – Build или F7 () – ошибки.

– Build – Execute или CTRL + F5 () – исполнение программы. Для переключения между проектами используйте Project – Set Active Project.

Структура программы:

```
#include <iostream.h>                    - процедуры препроцессора
int main()                               - функция main ()
{                                        - начало функции main ()
int x;                                   - объявление переменных
cout << "Введите число:\n";              - вывод данных
cin >> x;                                - ввод данных
for (int z=2; z<x; z++)
if (x % z == 0)
{
cout << "Это не простое число.\n";
return 0;
}
cout << "Это простое число.\n";
return 0;                                - нормальное завершение
}                                        - конец функции main ()
```

В C++ точка с запятой является признаком конца оператора.

Комментарии

// текст справа игнорируется компилятором

/* текст между игнорируется компилятором */

Таблица 8.1

Типы данных

Тип	Данные	Размер	Описание
char	символ	8 бит	Диапазон значений от 0 до 255. Может рассматриваться как число
int	Целые числа	16 бит	от –32768 до 32767
float	Числа с плавающей запятой одинарной точности	32 бита	Могут представляться как в фиксированном формате, например число π (3,14159), так и в экспоненциальном (7,56*103). Диапазон значений – +/-3,4E-38–3,4E+38
double	Числа с плавающей запятой двойной точности	64 бита	Диапазон значений от +/-1,7E-308 до +/-1,7E+308
void	указатели		Применяется в функциях, не возвращающих никакого значения
bool	логические (булевые)		Могут содержать только одну из двух констант: true или false

Ввод-вывод

Потоковый ввод-вывод консольному приложению обеспечивает стандартная библиотеку iostream.h:

```
cout << "Введите число:\n";          - вывод данных
cin >> x;                            - ввод данных
```

Форматирование вывода обеспечивают управляющие символы:

'\n' Новая строка (перевод строки)

'\t' Горизонтальная табуляция

'\\' Обратная косая

'\'' Апостроф

'\"' Кавычки

Библиотека stdio.h обеспечивает форматированный ввод-вывод:
#include <stdio.h>
printf("Vvedite znachenie: \n"); - вывод данных
scanf("%d",&i); - ввод данных
Управляющие символы форматирования (%...):
d – десятичное целое со знаком;
x – 16-ричное целое;
u – десятичное без знака;
e – число с плавающей запятой (экспоненциальное представление);
f – вещественное типа float;
c – символ;
s – строка;
o – восьмеричное без знака;
i – целое в любом формате.

Математические функции сведены в библиотеку math.h:
sin(x) – синус;
cos(x) – косинус;
tan(x) – тангенс;
acos(x) – арккосинус;
asin(x) – арксинус;
atan(x) – арктангенс;
atan2(x,y) – арктангенс x/y;
cosh(x) – гиперболический косинус;
sinh(x) – гиперболический синус;
tanh(x) – гиперболический тангенс;
acosh(x) – гиперболический арккосинус;
asinh(x) – гиперболический арксинус;
atanh(x) – гиперболический арктангенс;
hypot(a,b) – гипотенуза от двух катетов (a, b);
fabs(x) – абсолютная величина для double;
abs(x) – абсолютная величина для int;
pow(x,y) – степенная функция x^y;
pow10(x) – возведение числа 10 в степень x;
sqrt(x) – корень квадратный $\sqrt{x}$;
cbrt(x) – корень кубический $\sqrt[3]{x}$;

exp(x) – показательная функция e^x;

exp2(x) – степень «двух» 2^x;

expm1 – «экспонента без единицы» $e^x - 1$;

ldexp(x,a) – $x \cdot 2^a$ (где a – целое);
log(x) – натуральный логарифм;
log10(x) – десятичный логарифм;
log2(x) – логарифм по основанию 2;
round(x) – математическое округление до целого;
trunc(x) – отбрасывает дробную часть.
Здесь x – вещественное (double).

Задание:

1. Вычислить значение функции согласно своему варианту (варианты получить у преподавателя).

9. ОРГАНИЗАЦИЯ УСЛОВНЫХ ПЕРЕХОДОВ. РАЗВЕТВЛЯЮЩИЕСЯ АЛГОРИТМЫ

Оператор ветвления (условный оператор) «IF...ELSE...» в C++ называется просто оператором IF, и его общая структура:

```
if ( условие )
{
Оператор1;
...
...
ОператорN;
}
else
{
Оператор n+1;
...
...
Оператор n+m;
}
```

Часть ELSE в операторе IF необязательна. И если после условия IF стоит только один оператор, то можно опустить фигурные скобки.

При вычислении логических выражений 0 интерпретируется как FALSE, а любое значение, отличное от 0 - как TRUE. При построении логических выражений используются следующие операторы сравнения:

< – меньше
> – больше
<= – меньше или равно (не больше)
>= – больше или равно (не меньше)
== – равно
!= – не равно

Оператор выбора SWITCH передает управление одному из нескольких помеченных метками операторов в зависимости от значения целочисленного выражения. Метки оператора SWITCH имеют специальный вид:

```
switch (селектор)
{
```

case константа1:
[case константа2:]
. . .
[оператор;]
. . .
[break;]
[case константа_m:]
. . .
[case константа_n:]
[оператор;]
. . .
[break;]
[default:] [операторы]
}

1. Производится вычисление селектора (int-целое).

2. Полученное значение селектора поочередно сравнивается с константным выражением, следующим за case.

3. В случае если селектор равен константному выражению, стоящему за case, то управление передается оператору, с соответствующим оператором case.

4. break – обозначает немедленный выход из тела оператора SWITCH.

5. Варианту «ИНАЧЕ» соответствует оператор со словом default.

Задание:

Вычислить значение функции, согласно своему варианту (варианты получить у преподавателя).

10. ОРГАНИЗАЦИЯ ЦИКЛОВ

В языке С++ существует несколько способов организации циклических алгоритмов.

В том случае, когда необходимо выполнить определенное количество итераций используется конструкция FOR:

for (оператор 1; выражение 1; оператор 2; выражение 2);

Например, если необходимо выполнить набор операций для i от 0 до 10 с шагом 1, то запишется:

for (i=0; i < 10; i++) {набор операций};

Если количество повторений не задано, то используются циклы с предусловием:

while (выражение) оператор;

или циклы с постусловием:

do оператор (выражение);

Тело цикла DO – WHILE выполняется по крайней мере один раз в отличие от цикла WHILE.

Оператор BREAK осуществляет выход из тела цикла FOR, WHILE, DO-WHILE или оператора SWITCH, в котором он появился. Управление передается на первый оператор после цикла.

Оператор CONTINUE осуществляет переход на точку сразу за последним оператором тела цикла без выхода из цикла и дальнейшие итерации в цикле будут продолжаться.

Оператор RETURN завершает выполнение функции, в которой он задан, и возвращает управление в вызывающую функцию.

Задание:

1. Цикл FOR – вычислить сумму ряда, согласно варианту (варианты получить у преподавателя);

2. Цикл WHILE – вычислить значение заданной функции, согласно варианту;

3. Цикл DO-WHILE – Варианты заданий приведены в табл. 10.1.

Таблица 10.1

№ п/п	Задание
1	Организация цикла с подсчетом целых отрицательных чисел, вводимых с клавиатуры. Условие окончания цикла – ввод числа > 1000
2	Организация цикла с суммированием десятичных логарифмов целых чисел, вводимых с клавиатуры. Условие окончания цикла – ввод 0
3	Организация цикла с подсчетом количества четных чисел. Условие окончания цикла – ввод числа > 1000
4	Организация цикла с суммированием целых неотрицательных чисел, вводимых с клавиатуры. Условие окончания цикла – ввод числа < –100
5	Организация цикла, принимающего целые числа с клавиатуры с вычислением их среднего арифметического. Условие окончания цикла – ввод 0
6	Организация цикла, принимающего целые числа с клавиатуры с подсчетом кол-ва чисел, меньше 100. Условие окончания цикла – ввод числа 1000
7	Организация цикла с подсчетом количества нечетных чисел. Условие окончания цикла – ввод числа > 100
8	Организация цикла, принимающего целые числа с клавиатуры с вычислением суммы их синусов. Условие окончания цикла – ввод 0
9	Организация цикла, принимающего целые числа с клавиатуры с вычислением суммы их квадратов. Условие окончания цикла – ввод 0
10	Организация цикла с подсчетом количества четных чисел. Условие окончания цикла – ввод числа > 100
11	Организация цикла, принимающего целые числа с клавиатуры с подсчетом количества чисел, больше 100. Условие окончания цикла – ввод числа 1000
12	Организация цикла с подсчетом количества чисел, делящихся на 3. Условие окончания цикла – ввод числа > 1000

11. ОДНОМЕРНЫЕ И МНОГОМЕРНЫЕ МАССИВЫ

Массив – это набор данных одного и того же типа, которые собраны под одним именем. Каждый элемент массива определяется именем массива и порядковым номером элемента, называемый индексом (только целое число).

Объявление массива:

Тип <имя массива> [размер 1] [размер 2] ... [размер n];

Для одномерных массивов:

Тип <имя массива> [размер];

Тип данных – int, char, размер – количество элементов массива.

В языке Си индекс всегда начинается с 0.

Двумерные массивы можно представить как массив одномерных массивов.

Существует несколько способов инициализации массивов:

1) float arr [6] = {1.0, 5.2, 8.3, 4.9, 8.5, 96}; одномерный массив int aq [3] [5] = {7, 2, 5, 4, 1, 6, 24, 8, 9, 10, 101, 22, 83, 14, 175}

2) b[0] [0] = 78; b [0] [1] = 89; b [0] [2] = 5; b [0] [3] = 99; b [0] [4] = 7; b [0] [5] = 23; b [0] [6] = 0; и т. д.

Многомерные массивы, в том числе и двумерные, можно инициализировать, рассматривая их как массив массивов.

Инициализации: int a [3] [5] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 101, 102, 103, 104, 105}; и int a [3] [5] = {{10, 20, 30, 40, 50}, {60, 70, 80, 90, 100}, {101, 102, 103, 104, 105}}; эквиваленты.

Символьные массивы можно инициализировать как обычные:

char str [15] = {‘V’, ‘i’, ‘s’, ‘u’, ‘a’, ‘l’ ‘C’, ‘+’, ‘+’}; а могут как строка: char str [15] = ‘VisualC++’;

Возможно объявление массива без указания размера:

int mas [ ] = {1, 2, 3, 4, 5,1, 2};

Компилятор сам определяет количество элементов массива. Отдельное место в языке Си занимают символьные массивы. Во многих языках программирования существует тип – строка символов. В Си++ этот тип данных отсутствует, и для реализации строк используются одномерные массивы.

Задание:

Выполнить программу согласно своему варианту:

1. Найти сумму элементов массива с четными индексами.

2. Найти среднее арифметическое положительных элементов массива.

3. Удвоить каждый элемент массива. Полученный массив напечатать.

4. Найти произведение элементов массива с нечетными индексами.

5. Переписать первые элементы каждой строки матрицы A(3,3), которые больше 10, в массив B.

6. Задан двумерный массив A(3,3). Если на главной диагонали стоит нуль, то соответствующую строку заменить единицами.

7. Задан двумерный массив C(4,4). Если максимальный элемент массива находится на главной диагонали, то все элементы главной диагонали сделать равными наибольшему.

8. Задан двумерный массив A(5,5). Заменить все элементы 1 и 5 строки, которые больше 10, на 5.

9. Задан двумерный массив C(4,4). Поменять местами максимальный элемент каждой строки с первым элементом соответствующей строки.

10. Сформировать одномерный массив из 20 случайных целых чисел в диапазоне от –30 до +30. Вывести его на экран. Сформировать новый массив, каждый элемент которого меньше на 10 элементов заданного массива.

11. Задан двумерный массив C(4,4). Отсортировать все элементы первой строки в убывающем порядке.

12. Задан двумерный массив A(4,4). Отсортировать все элементы второго столбца в возрастающем порядке.

12. УКАЗАТЕЛИ

Указатель – это переменная или константа, содержащая адрес другой переменной.

Объявление указателя включает базовый тип, * и имя переменной:

тип *имя;

Базовый тип указателя определяет тип переменной, на которую указывает указатель. Значение указателя – это целое число без знака длиной 4 байта. Оно сообщает, где расположена переменная.

Таким образом, в указателе очень важным является базовый тип, т.к. он определяет, сколько байтов занимает переменная указатель. Если int – 2 байта, char – один байт и т. д.

Имеется два специальных оператора для работы с указателями – * и &. Обе эти операции являются унарными, т. е. они имеют один операнд, перед которыми они ставятся.

Оператор & возвращает адрес операнда:

a = &x; //помещает адрес переменной *x* в *a*.

Данный адрес соответствует внутреннему положению переменной в памяти компьютера. Он не изменяет значение *x*. Операцию & можно рассматривать как «взятие адреса». Т. е. оператор *a* = &x; можно прочитать как «*a* получает адрес *x*».

Оператор * возвращает значение переменной, находящейся по указанному адресу. Например, если *p* содержит адрес памяти переменной X, то

```
qw=*p; // *qw получает значение по адресу p
 и поместит значение переменной X в переменную qw
Например,
#include <stdio.h>
int main(void)
{
int x, qw;
int *p;
x = 10; /* x присваивается 10 */
p = &x; /* p получает адрес x */
qw = *p; /* qw с помощью p присваивается значение x */
printf("%d", qw); /* выводит 10 */
return 0;
}
```

Данная программа выводит значение 10 на экран.

К указателям могут применяться только две арифметические операции: сложение и вычитание. Допустим, что *p*1 – это указатель на целое, содержащий значение 1000, и будем считать, что целые имеют длину 2 байта. После выражения

*p*1 ++;

содержимое *p*1 станет 1002, а не 1001! Каждый раз при увеличении *p*1 указатель будет указывать на следующее целое. Это справедливо и для уменьшения. Например:

*p*1 --;

приведет к тому, что *p*1 получит значение 998, если считать, что раньше было 1000.

Каждый раз, когда указатель увеличивается, он указывает на следующий элемент базового типа. Каждый раз, когда уменьшается – на предыдущий элемент. В случае указателей на символы это приводит к «обычной» арифметике. Все остальные указатели увеличиваются или уменьшаются на длину базового типа. Кроме того, можно добавлять или вычитать из указателей целые числа.

Указатели одного и того же типа могут использоваться в операции присваивания, как любые другие переменные.

```
# include <stdio.h>
main ()
{
int x = 0 ; int *p, *qw; p = &x;
qw = p;
printf("%p", p); /* печать адреса p */
printf("%p", qw ); /* печать адреса qw */
printf("%d %d", x., *qw); /*печать величины x и величины по адресу qw */
}
```

где, %*p* – печать адреса памяти в шестнадцатеричной системе счисления.

Доступ к каждому элементу массива осуществляется по его индексу. Элементы массива нумеруются начиная с 0 и занимают последовательные ячейки памяти. При этом они размещаются таким образом, что самый последний индекс возрастает быстрее. Это в случае двумерного массива означает, что он будет записываться построчно: строка за строкой. Поскольку указатели указывают адрес ячейки, то

между массивами и указателями существует тесная связь. Имя массива – это указатель на его первый элемент. По существу массив можно рассматривать как индексированный указатель. Доступ к элементам массива осуществляется по номеру индекса. При этом приращение индекса на единицу вызывает перемещение указателя на число байт, соответствующее объекту данного типа: для целых чисел на 2 байта, для действительных – на 4 байта и т. д.

Объявления int arr[] и int *arr идентичны по действию: оба объявляют arr указателем.

Пример

```
int arr[10];
int *p;
p = arr; // присваивает адрес указателю
// следующие операции приводят к одному результату:
arr[2] = 10;
*(p + 2) = 10; // прибавить 2 к первому элементу: *p + 2;
```

Таким образом, указатели

– предоставляют способ, позволяющий функциям модифицировать передаваемые аргументы,

– используются для поддержки системы динамического выделения памяти,

– их использование может повысить эффективность работы некоторых подпрограмм.

Задание:

1. Число А (по варианту лаб. раб. № 2) из 10 с/с перевести в двоичную, затем в шестнадцатеричную и восьмеричную с/с. Записать в 32-разрядную сетку с учетом знака числа и смещенного порядка.

2. Исходное число записать по адресу ячейки памяти, заданному с помощью указателя. Считать содержимое в различных системах счисления и проверить результат выполнения расчетов первой части задания.

3. Число В, представленное в 2 с/с побайтно, перевести в 10 с/с. Результат записать по адресу ячейки памяти, с использованием указателя. Прибавить к содержимому число А и результат вывести в 10 и 16 с/с.

3. Согласно варианту восстановить смещенный порядок Р по правилу (см. лаб. раб. №2).

Спецификатор	Формат
%i, %d	int
%c	char
%f, %F	double, float
%s	Символьная строка
%o	Восмеричное число
%x, %X	Шестнадцатеричное число
%li	long int
%lf	long double
%%	Символ '%'
%e, %E	Экспоненциальное представление числа
%g, %G	Использование наиболее краткой записи среди форматов %e и %f
%p	Выводит указатель

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Новиков, Ю. В. Основы микропроцессорной техники: Учебное пособие / Ю. В. Новиков, П. К. Скоробогатов. – М.: БИНОМ. ЛЗ, ИНТУИТ.РУ, 2012. – 357 с.

2. Новиков, Ю. В. Основы микропроцессорной техники: курс лекций / Ю. В. Новиков, П. К. Скоробогатов. – М.: ИНТУИТ.РУ, 2003. – 440 с.

3. Новиков, Ю. В. Основы микропроцессорной техники / Ю. В. Новиков, П. К. Скоробогатов. – М.: Бином. Лаборатория знаний, 2009. – 357 с.

4. Новожилов, О. П. Основы микропроцессорной техники : учебное пособие: в 2-х т. / О. П. Новожилов. – М.: ИП РадиоСофт, 2011. – 336 с.

5. Новожилов, О. П. Основы микропроцессорной техники : учебное пособие: в 2-х т. / О. П. Новожилов. – М.: РадиоСофт, 2007. – 336 с.

6. Новожилов, О. П. Основы микропроцессорной техники : учебное пособие: в 2-х т. / О. П. Новожилов. - М.: РадиоСофт, 2007. – 432 с.

7. Остриков, А. Н. Основы микроэлектроники и микропроцессорной техники : учебное пособие / А. Н. Остриков, М. И. Слюсарев, Е. Ю. Желтоухова. – СПб.: Лань, 2013. – 496 с.

8 Ершова, Н. Ю. Микропроцессоры / Н. Ю. Ершова, О. Н. Ивашеноков, С. Ю. Курсковю – Питер, 2002. – 265 с.

9. Москаленко, А. А. Микропроцессоры : методическое пособие / А. А. Москаленко, З. И Кононенко, А. Р. Околов. – Минск: БНТУ, 2012. – 61 с.

10. Литвиненко Н. А. Технология программирования на С++. – СПб.: – БХВ-Петербург, 2010. – 281 с.

11. Хортон Айвор. Visual C++ 2010. Полный курс.: пер. с англ. – М.: ООО «И. Д. Вильямс», 2011. – 1216 с.

12. Шилдт Г. С++ Базовый курс. 3-е изд. пер. с англ. – М.: ООО «И. Д. Вильямс», 2010. – 624 с.

Учебное издание

МАТРУНЧИК Юлия Николаевна

МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ УПРАВЛЕНИЯ

Лабораторный практикум
для студентов специальности 1-53 01 01 «Автоматизация технологических процессов и производств (по направлениям)»

Редактор *Е. В. Герасименко*
Компьютерная верстка *Е. А. Беспанской*

Подписано в печать 17.03.2020. Формат 60×84 $^{1}/_{16}$. Бумага офсетная. Ризография.
Усл. печ. л. 3,84. Уч.-изд. л. 3,00. Тираж 100. Заказ 905.

Издатель и полиграфическое исполнение: Белорусский национальный технический университет.
Свидетельство о государственной регистрации издателя, изготовителя, распространителя печатных изданий № 1/173 от 12.02.2014. Пр. Независимости, 65. 220013, г. Минск.