Министерство науки и высшего образования
Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра телевидения и управления (ТУ)

В.А. Кормилин

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Учебно-методическое пособие по организации практических занятий и самостоятельной работы

2019

Кормилин В.А.
Вычислительная техника: Учебно-методическое пособие по организации практических занятий и самостоятельной работы. Томск: Томский государственный университет систем управления и радиоэлектроники (ТУСУР), 2019. 41 с.

Учебно-методическое пособие предназначено для студентов радиотехнических направлений подготовки ТУСУРа, обучающихся на всех формах обучения и содержит учебный материал и методические указания для организации практических занятий и самостоятельной работы при изучении дисциплины.

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

При разработке заданий на практические занятия и для организации самостоятельной работы учитывалась необходимость формирования и оценки достижения компетенций, связанных с данной дисциплиной в рабочем учебном плане.

Формируемые знания, умения и навыки связаны со следующими компетенциями:

ОПК-4 - способность применять современные компьютерные технологии для подготовки текстовой и конструкторско-технологической документации с учетом требований нормативной документации;

ПКС-1 – способность выполнять расчет и проектирование элементов и устройств инфокоммуникационных систем в соответствии с техническим заданием с использованием средств автоматизации проектирования.

1 ПРАКТИЧЕСКОЕ ЗАНЯТИЕ. ЗНАКОМСТВО С СИСТЕМАМИ СЧИСЛЕНИЯ

1.1 Цель занятия

Целью практического занятия является практическое знакомство с системами счисления, используемыми в вычислительной технике, изучение способов преобразования между системами счисления и первичное изучение методов выполнения арифметических операций в разных системах счисления.

1.2 Позиционный принцип

Основанием системы счисления **p** будем называть число различных цифр-символов, используемых для представления любого числа в данной системе счисления. Любое число представляется в виде последовательности цифр, разделенных запятой на две группы. Группа цифр, расположенных левее запятой, образует целую часть числа. Правее запятой расположена дробная часть числа.

Каждая цифра числа занимает в нем определенную позицию, называемую разрядом. Разрядам приписываются различные весовые коэффициенты. Коэффициенты соответствуют различным целым степеням, в которые возводится основание системы счисления. Цифры данного числа представляются в виде символов некоторого счетного алфавита, обозначающих целые числа, находящиеся в диапазоне $0 < a_j < (p-1)$.

Используя **позиционный принцип** можно представить любое число N в любой системе счисления с основанием p:

$$N = a_n p^n + a_{n-1} p^{n-1} + \cdots + a_0 p^0 + a_{-1} p^{-1} + a_{-2} p^{-2} + \cdots + a_{-m} p^{-m}$$

В вычислительной технике широкое распространение получили несколько систем счисления. Рассмотрим наиболее употребительные системы счисления.

1.3 Десятичная система счисления

Десятичная система счисления, знакомая всем со школы, имеет основание **p=10**. Для представления любого числа достаточно десяти цифр: 0, 1, 2, 3,…,9, из которых и составляются разряды числа и которые составляют алфавит десятичной системы счисления. Цифры целой части числа имеют вес 10^0, 10^1, 10^2, …, а дробная часть имеет коэффициенты вида 10^{-1}, 10^{-2}, 10^{-3}, и т.д.

Таким образом, число 9875,432 в десятичной системе счисления можно представить в следующем виде:

$$9875{,}432 = 9\cdot 10^3 + 8\cdot 10^2 + 7\cdot 10^1 + 5\cdot 10^0 + 4\cdot 10^{-1} + 3\cdot 10^{-2} + 2\cdot 10^{-3}$$

1.4 Двоичная система счисления

Основанием двоичной системы является **p=2**, а для представления разрядов чисел используются два символа: 0 и 1 (алфавит системы).

Укажем пример числа в двоичной системе: 101101,012. Позиционный принцип дает следующее значение данного числа:

$$N = 1\cdot 2^5 + 0\cdot 2^4 + 1\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 0\cdot 2^{-1} + 1\cdot 2^{-2} = 45{,}25_{10}$$

Вес дробных разрядов, таким образом, равен 0,5, 0,25, 0,125 и далее как дробь $(1/2)^n$. Коэффициенты для разрядов целой части числа равны 1, 2 , 4, 8, 16, 32 и т.д.

1.5 Восьмеричная система счисления

Основание системы счисления **p=8**. В каждом разряде может использоваться один из восьми символов: 0, 1, 2, 3, 4, 5, 6 или 7. Пример числа в данной системе счисления имеет вид: 72645,278 . Значение числа равно:

$$N = 7\cdot 8^4 + 2\cdot 8^3 + 6\cdot 8^2 + 4\cdot 8^1 + 5\cdot 8^0 + 2\cdot 8^{-1} + 7\cdot 8^{-2} = 30117{,}359_{10}$$

Вес дробных разрядов в данной системе равен 1/8, 1/64, 1/256 и далее как дробь $(1/8)^n$. Коэффициенты для веса разрядов в целой части числа равны 1, 8, 64, 256, 4096 и т.д.

1.6 Шестнадцатеричная система счисления

Основание системы **p=16**. Для обозначения цифр каждого разряда требуется 16 различных символов. Таким образом, алфавит шестнадцатеричной системы содержит 16 символов. Цифры от 0 до 9 соответствуют цифрам из десятичной системы счисления. Для обозначения недостающих шести символов, соответствующих значениям 10, 11, 12, 13, 14 и 15 используются английские буквы, соответственно, A, B, C, D, E и F. Рассмотрим пример числа 2F34,C816.

$$N = 2\cdot 16^3 + 15\cdot 16^2 + 3\cdot 16^1 + 4\cdot 16^0 + 12\cdot 16^{-1} + 8\cdot 16^{-2} = 12084{,}78125_{10}$$

В данной формуле числом символ F заменен числом 15, а символ C – числом 12.

1.7 Обозначение системы счисления

При работе только в одной системе счисления проблем для обозначения системы числа не возникает. Труднее обстоит дело при возможности одновременного использования нескольких систем. Одинаковое число в разных системах счисления будет иметь разные значения. Поэтому появилась потребность обозначать систему счисления числа.

Чтобы отличать десятичные числа, можно указывать индекс 10 .Запись будет выглядеть как 24687_{10}. Использование индексов очень неудобно. Было предложено для обозначения системы счисления применять буквы, указываемые после самого числа. В вычислительной технике принято, что числа, не помеченные никакой буквой, или имеющие в конце букву D (сокращение от

DECIMAL) являются десятичными числами. Например, правильно десятичное число можно записать, как 1101D.

Для обозначения двоичной системы числа можно применять индекс 2, как показано выше, но рекомендуется использовать букву B (сокращение от BINARY), указываемую после числа. Например, двоичное число можно записать как 011010110B.

Для обозначения восьмеричного числа можно использовать индекс 8. Рекомендовано после такого числа указывать букву O (сокращение от OCTAL). В связи с возможностью совпадения начертания буквы O с цифрой 0, восьмеричные числа на практике обозначают буквой Q. Пример правильного обозначения чисел 163527O, 267453Q.

Индекс 16 позволит отличать числа шестнадцатеричной системы от любой другой. Но реально используется обозначение такого числа буквой H. Буквенное обозначение H взято от английского слова HEXADECIMAL. Пример числа в данной системе имеет вид 4FA6H. Кроме этого, если шестнадцатеричное число начинается с буквы, то принято добавлять в начале цифру 0. Другой пример запишем как: 0B3A4H. При этом соблюдается соглашение, которое гласит: числа должны начинаться с цифры, а не буквы. Этим устраняется схожесть в обозначении шестнадцатеричных чисел и обозначении служебных имен и меток.

1.8 Правила преобразования между системами счисления

Сформулируем основные правила преобразования из любой системы счисления в десятичную и обратно.

Правило 1 перевода в десятичную систему счисления из произвольной системы счисления. Старшую цифру числа умножаем на основание системы счисления **p**, возведенное в степень, равное уменьшенному на единицу номеру цифры в числе. К произведению прибавляем следующую цифру, умноженную на возведенную в соответствующую степень основание счисления. Умножение и суммирование повторяем до исчерпания всех цифр целой части числа.

Правило 2 перевода в произвольную систему счисления из десятичной целого числа.

Делим целое десятичное число на основание выбранной системы счисления p. Запоминаем остаток, как значение самой младшей цифры преобразованного числа. Делим полученное частное на p. Снова фиксируем остаток, как значение следующей, более старшей цифры. Повторяем процесс деления до получения частного, меньшего, чем p. Последовательно получаемые в процессе деления остатки являются цифрами искомого числа, от младшей к старшей цифре.

Указанные правила позволяют легко преобразовывать числа между десятичной и двоичной системами, между десятичной и восьмеричной системами и т.д.

1.9 Задание на практическое занятие

1. Найдите ваше персональное число. Для этого запишите год своего рождения, просуммируйте число с номером месяца рождения и прибавьте число рождения.
2. Преобразуйте ваше персональное число в двоичную систему счисления.
3. Преобразуйте ваше персональное число в восьмеричную систему счисления.
4. Преобразуйте ваше персональное число в шестнадцатеричную систему счисления.
5. Запишите результаты с правильным обозначением чисел и представьте их для проверки преподавателю.

1.10 Контрольные вопросы

1. Что такое основание системы счисления?
2. Что такое алфавит системы счисления?
3. Как обозначается число в десятичной системе счисления?
4. Как обозначается число в двоичной системе счисления?
5. Как обозначается число в восьмеричной системе счисления?
6. Как обозначается число в шестнадцатеричной системе счисления?

2 ПРАКТИЧЕСКОЕ ЗАНЯТИЕ. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ С ЧИСЛАМИ В ДВОИЧНОЙ СИСТЕМЕ СЧИСЛЕНИЯ

2.1 Цель занятия

Целью практического занятия является практическое изучение методов выполнения различных арифметических операций в разных системах счисления: суммирование, вычитание, умножение и деление.

2.2 Арифметические операции в прямом коде

Основными арифметическими операциями являются операции суммирования, вычитания, умножения и деления чисел. Со школьной скамьи мы знаем, что операции умножения и деления можно свести к операциям суммирования и вычитания. В свою очередь, операцию вычитания можно заменить операцией суммирования с отрицательным числом.

Поэтому будем считать операцию суммирования основной арифметической операцией, при условии, что числа могут быть как положительными, так и отрицательными.

При записи кода числа знак плюс принято обозначать цифрой 0, а знак минус – цифрой 1. Если в числе необходимо отобразить знак, то числа будем называть знаковыми, иначе без знаковыми. Для двоичных знаковых чисел старший разряд числа называется знаковым. Для простоты изложения мы будем знаковый бит заключать в квадратные скобки и рассматривать все числа, как целые.

Сложение положительных чисел. Рассмотрим операцию сложения на примере двоичных положительных чисел. При этом основным правилом сложения двоичных чисел является правило побитного сложения. Всего возможно 4 сочетания битов. Рассмотрим результаты сложения пары битов для всех возможных сочетаний.

Первый бит	1	0	1	0
Второй бит	1	1	0	0
Сумма	0	1	1	0
перенос	1			

Здесь приведены все возможные сочетания суммы двух битов. При появлении переноса, он учитывается при суммировании других, более старших битов. Рассмотрим пример сложения многоразрядных чисел:

Переносы	111
Первое слагаемое	[0]01011
Второе слагаемое	[0]01110
Сумма	[0]11001

При суммировании чисел возникают переносы, которые суммируются с последующими разрядами.

Умножение двоичных чисел. При выполнении операции умножения нужно определить знак результата и найти абсолютное значение произведения.

Знаковый разряд произведения можно определить суммированием битов знаковых разрядов по модулю 2. Эта операция обозначается как ⊕. Действительно, при равенстве знаков сомножителей (0 и 0 или 1 и 1) произведение будет иметь положительный знак (обозначаемый как 0), а сумма по модулю 2 дает нуль, т.е. 0 ⊕ 0 = 0 и 1 ⊕ 1 = 0. При несовпадении знаков произведение будет иметь знак минус (т.е. 1), а 0 ⊕ 1 = 1 и 1 ⊕ 0 = 1.

Определение абсолютного значения произведения выполняется без учета знаков сомножителей. Наиболее просто перемножение выполняется методом сдвига множимого влево и суммированием частичных произведений. Приведем пример умножения:

				1	0	0	1	1
	*				1	0	1	1
				1	0	0	1	1
	+		1	0	0	1	1	
	+	0	0	0	0	0		
	+	1	0	0	1	1		
Итог		**1**	**1**	**0**	**1**	**0**	**0**	**0**	**1**
Переносы				1	1	1	1	1	

Как видно из примера, в процессе выполнения операции умножения суммируются сдвинутые на заданное число разрядов частичные произведения (произведения множимого на цифры разрядов множителя). Такие частичные произведения равны либо сдвинутому влево множимому, либо нулю. Это определяется соответствующим битом множителя. Например, второй бит множителя равен 1, значит, сдвинутое влево на 1 разряд множимое суммируется с промежуточной суммой.

В цифровых устройствах процессу суммирования частичных произведений придают последовательный характер, что и объясняется появлением термина «промежуточная сумма».

Нетрудно догадаться о причине появления среди частичных произведений нулевых членов. Они являются результатом умножения множимого на нулевой бит множителя. В общем случае, процесс суммирования можно начинать с частичного произведения, получаемого умножением множимого на старший, либо младший биты множителя. Этим определяется только направление сдвига частичного произведения относительно накапливаемой суммы (вправо или влево).

При умножении целых чисел результат, в общем виде, требует наличия числа разрядов, равных сумме числа разрядов сомножителей. Так, перемножение 5-ти битового числа на 4-х битовое дает 9-ти битовый результат.

Деление двоичных чисел. При делении двоичных чисел можно использовать различные алгоритмы. Один из простых алгоритмов предполагает вычитание делителя из делимого и подсчитывание количества вычитаний до получения отрицательного результата. Существенный недостаток данного алгоритма: зависимость времени выполнения операций от соотношения делимого и делителя.

Рассмотрим алгоритм, который называется делением целых чисел с восстановлением остатка. Правильные результаты получаются, если число разрядов делимого больше числа разрядов делителя. Рассмотрим пример деления числа N1 = 10010111_2 = 151_{10} на N2 = 1011_2 = 11_{10}. Обозначим бит переноса в фигурных скобках. Совместим старшие биты делимого и делителя и начинаем вычитать.

			1	0	0	1	0	1	1	1	
		-	1	0	1	1					Вычитаем делитель
0	←	{1}	1	1	1	0	0	1	1	1	есть заем, восстанавливаем остаток
			1	0	0	1	0	1	1	1	восстановленный остаток
		-		1	0	1	1				Сдвиг делителя вправо. Вычитаем
1	←	{0}	0	0	1	1	1	1	1	1	Нет заема, продолжаем
					1	0	1	1			Сдвиг делителя вправо. Вычитаем
1	←	{0}	0	0	0	1	0	0	1	1	Нет заема, продолжаем
						1	0	1	1		Сдвиг делителя вправо. Вычитаем
0	←	{1}	1	1	1	1	1	1	0	1	есть заем, восстанавливаем остаток
	←		0	0	0	1	0	0	1	1	восстановленный остаток
							1	0	1	1	Сдвиг делителя вправо. Вычитаем
1	←	{0}	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	Нет заема. Завершаем деление, берем остаток

Процесс прекращаем при достижении исходного значения делителя. Биты частного равны 01101_2 , а остаток равен 1000_2. Это правильный результат деления $151_{10}/11_{10}=13_{10}$ и 8_{10} в остатке. Биты частного получились фиксацией инверсии бита переноса, начиная с первого, как самого старшего бита результата.

Опишем алгоритм деления.

1. Задаем начальное значение частного равным 0. Совмещаем делитель со старшими разрядами делимого.
2. Вычитаем из делимого делитель.
3. Определяем текущий бит частного, начиная со ставшего. Текущий бит частного оценивается по состоянию бита переноса. Если бит переноса равен 1, нужно в бите частного запомнить 0, а остаток восстановить (к остатку прибавить делитель). Если перенос равен 0, то в результате фиксируем 1, и ничего восстанавливать не нужно.
4. Сдвигаем делитель на разряд вправо, заполняя нулями свободные левые биты. Повторяем вычитание уже сдвинутого делителя из имеющегося остатка.
5. Оцениваем бит частного, аналогично шагу 3.

6. Если делитель еще не сравнялся со своим исходным значением (младший бит делителя не совместился с младшим битом делимого), идем на наг 4, иначе 7.

7. Останавливаем процесс деления.

Данный алгоритм требует значительно меньшее число итераций, чем простой подсчет количества вычитаний делителя из делимого. Недостатком алгоритма является необходимость восстанавливать остаток при получении отрицательного результата вычитания. Некоторый выигрыш в количестве операций дает алгоритм деления без восстановления остатка.

Можно предложить и другие алгоритмы деления, например, составление таблицы всех возможных произведений. Количество строк будет равно количеству значений первого сомножителя, а количество столбцов – количеству значений второго сомножителя. Элементы таблицы равны произведениям сомножителей.

Деление нацело происходит по нахождению номера столбца, в котором находится ближайшее к делимому число в строке с номером, равным делителю. Такой метод деления обеспечивает самую высокую скорость деления. Существенный недостаток метода – очень большой размер таблицы.

2.3 Дополнительный код

При сложении положительных чисел трудностей обычно не возникает. Труднее складывать числа, имеющие знаки. Для чисел со знаком операция сложения выполняется только при равенстве знаков. Для чисел с разными знаками вместо сложения приходится применять операцию вычитания. Фактически, вид операции зависит от сочетания знаковых разрядов.

Для упрощения таких операций рекомендуется использовать дополнительный код. Для пояснения принципа использования дополнительного кода, рассмотрим простой пример вычитания положительных десятичных чисел. Пусть N1 = 756, а N2 = 279. Запишем знак чисел N1=[0]756, а N2=[0]279. Необходимо вычислить разность N1- N2. Заменим операцию вычитания операцией сложения с отрицательным числом, т.е. изменим знак второго числа. Тогда N1=[0]756, а N2=[1]279 .

Выполнение такого сложения по школьным правилам потребует последовательности действий с заемами из старших разрядов.

```
  [0]756
+ [1]279
---------
  [0]477
```

В цифровом устройстве не обязательно предусматривать такую последовательность действий. Преобразуем отрицательное число в дополнительный код следующим образом: во всех разрядах, кроме знакового, заменим каждую цифру дополнением до девяти к значению этой цифры (при этом каждую цифру нужно как бы вычесть из цифры 9). Цифра 2, вычтенная из 9,

превратится в 7, цифра 7 – аналогично в 2. Дополнение до 9 цифры 9 дает 0. Получим N2доп-1=[1]720. В заключении к числу приплюсуем единицу в младший разряд. Мы получили дополнительный код отрицательного числа N2доп=[1]721. Почему дополнение до 9? Мы фактически вычли наше отрицательное число из максимального числа, которое можно получить при заданном числе разрядов. А это максимальное число ровно на единицу больше числа, в котором все разряды равны 9. Цифра 9 является наибольшей цифрой в данной системе счисления. Отсюда и понятен способ получения дополнительного кода с вычитанием из 9 и прибавлением 1 в заключении.

Теперь будем выполнять операцию суммирования, не обращая внимания на знаки чисел и результата.

```
          [0]756
         +[1]721
          [0]477
перенос     1  1
```

В примере суммируются и двоичные цифры знаковых разрядов. При этом отбрасывается перенос, возникающий из знаковых разрядов. С удовлетворением замечаем, что получен правильный результат операции.

В двоичной системе в дополнительном коде потребуется дополнение каждого бита до единицы (помните, максимальная цифра в данной системе счисления). Это можно выполнить простым инвертированием (замена 0 на 1, а 1 на 0) всех битов, кроме знакового. Не забывайте в заключении прибавить единицу к младшему разряду числа. Например, N=[1]0010110 преобразуется в Nдоп=[1]1101010. Обратное преобразование выполняется по тому же правилу.

Рассмотрим примеры сложения двоичных чисел в дополнительном коде. Пусть первое слагаемое N1 = [0]01101002 = 5210, имеет положительный знак, а второе слагаемое N2 = [1]01011012 = -4510 задано со знаком минус. Преобразуем второе число в дополнительный код. Для этого выполним инверсию числа, за исключением знакового бита, и прибавим к числу единицу. Получим N2доп. = [1]10100112. Выполняем саму операцию сложения:

```
          [0]0110100
          [1]1010011
          [0]0000111
переносы    1 111
```

Не забываем, что возникающий из знаковых разрядов перенос нужно отбросить. Получили число [0]00001112 = 710 . Видим, что это правильный результат сложения исходных чисел с учетом их знаков.

Изменим на обратные знаки слагаемых из предыдущего примера. Первое слагаемое N1 = [1]01101002 = -5210 теперь отрицательно, а второе слага-

емое N2 = [0]01011012 = +4510 имеет положительный знак. Ожидаемый результат сложения чисел равен [1]00001112 = -710. Преобразуем отрицательное число N1 в дополнительный код N1доп = [1]10011002. Выполняем сложение чисел.

```
[1]1001100
[0]0101101
[1]1111001
переносы  1  1
```

Ответ получился отрицательным, но что-то не похож на ожидаемый результат! Да ведь он, наверно, в дополнительном коде! Преобразуем число из дополнительного кода по известному алгоритму: инверсия всех битов, за исключением знакового, и суммирование с единицей. Получим результат СУММА=[1]0000111. Да, это то, что ожидалось!

Подведем некоторый итог. При использовании дополнительного кода нужно заранее преобразовать числа. Дополнительный код положительного числа равен самому числу, а дополнительный код отрицательного числа получается инвертированием всех битов числа, за исключением знакового разряда, и увеличением на единицу полученного числа. При выполнении операции сложения в дополнительном коде нужно отбрасывать единицу переноса из знакового бита. Причем положительный результат суммирования равен самому числу, а отрицательный результат получается в дополнительном коде.

Таким образом, применение дополнительного кода позволяет отказаться от операции вычитания с заменой ее на операцию суммирования в дополнительном коде. Этот прием широко используется при знаковых вычислениях в микропроцессорах.

2.4 Задание на практическое занятие

1. Запишите четыре цифры своего года рождения. Преобразуйте эти цифры в двоичную систему счисления.
2. Преобразуйте две цифры своего дня рождения в двоичную систему.
3. Выполните суммирование этих чисел.
4. Выполните умножение этих чисел.
5. Преобразуйте две цифры месяца своего рождения в двоичную систему счисления.
6. Выполните вычитание из года рождения цифр месяца в дополнительном коде.
7. Запишите результаты и представьте их для проверки преподавателю

2.5 Контрольные вопросы

1. Как обозначается знак двоичного числа?
2. Как выполняется операция умножения двоичных чисел?
3. Как при перемножении заранее определить количество разрядов произведения?

3 ПРАКТИЧЕСКОЕ ЗАНЯТИЕ. ЗНАКОМСТВО С МЕТОДАМИ РАЗРАБОТКИ ПРОГРАММ ДЛЯ МИКРОПРОЦЕССОРОВ

3.1 Цель занятия

Целью практического занятия является начальное изучение методов разработки программ для микропроцессоров, знакомство с алгоритмами, изучение способов алгоритмизации различных прикладных задач.

3.2 Алгоритмы программ

В процессе разработки программ при программировании микропроцессоров необходимо представление решаемой задачи в виде алгоритма.

АЛГОРИТМ – (от латинской формы имени арабского математика Аль Хорезми) - совокупность и последовательность действий, система правил для решения конкретной задачи, последовательность проведения вычислительных операций, способ нахождения искомого результата.

Алгоритм представляет собой последовательное изложение шагов-действий, направленных на решение конкретной прикладной задачи. Алгоритм практически не зависит от самого микропроцессора, его системы команд.

Таким образом, алгоритмом называется точная инструкция исполнителю в понятной для него форме, определяющая процесс достижения поставленной цели на основе имеющихся исходных данных за конечное число шагов.

Основными свойствами алгоритмов являются:

1. Универсальность (массовость) - применимость алгоритма к различным наборам исходных данных.

2. Дискретность - процесс решения задачи по алгоритму разбит на отдельные действия.

3. Однозначность - правила и порядок выполнения действий алгоритма имеют единственное толкование.

4. Конечность - каждое из действий и весь алгоритм в целом обязательно завершаются.

5. Результативность - по завершении выполнения алгоритма обязательно получается конечный результат.

6. Выполнимость - результат алгоритма достигается за конечное число шагов.

Алгоритм считается правильным, если его выполнение дает правильный результат. Соответственно алгоритм содержит ошибки, если можно указать такие допустимые исходные данные или условия, при которых выпол-

нение алгоритма либо не завершится вообще, либо не будет получено никаких результатов, либо полученные результаты окажутся неправильными.

Из всего многообразия различных типов алгоритмов будем рассматривать управляющие алгоритмы, генерирующие различные управляющие воздействия на основе данных, полученных от внешних процессов, которыми алгоритмы управляют.

3.3 Способы изображения алгоритмов программ

Алгоритм можно изобразить различными способами:

- в текстовом виде, когда алгоритм описывается на человеческом языке;
- в графическом виде, когда алгоритм описывается с помощью набора графических изображений;
- в символьном виде, когда алгоритм описывается с помощью набора символов или команд конкретного языка программирования.

Наиболее простым и интуитивно понятным представляется текстовый вид представления алгоритма. В этом случае разработчик с помощью обычных слов формулирует последовательность действий.

В качестве примера можно рассмотреть рецепт приготовления блюда – вареного картофеля.

1. Начало.
2. Наполнить кастрюлю холодной водой.
3. Приготовить 5 картофелин.
4. С помощью ножа очистить картофелины от кожуры.
5. Положить очищенные картофелины в кастрюлю с водой.
6. Включить плиту.
7. Поставить кастрюлю с картофелем на плиту.
8. Вода закипела?
   Нет? - возвращаемся на шаг 8.
   Да?- переходим на шаг 9.
9. Ждем 1 минуту
10. Протыкаем вареную картофелину вилкой.
11. Вилка втыкается легко?
    Нет? - идем на шаг 9.
    Да? –переходим на шаг 12.
12. Сливаем воду.
13. Конец алгоритма.

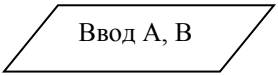
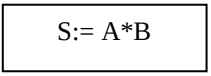
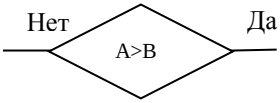
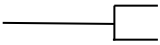
Другим примером описания алгоритма в текстовом виде является случай объяснения маршрута прохожему, типа «Как пройти на почту?»

Другим способом и более наглядным представлением алгоритма является графический способ. Существует несколько способов графического

описания алгоритмов. Наиболее широко используемым на практике графическим описанием алгоритмов является использование блок-схем. Несомненное достоинство блок схем – наглядность и простота записи алгоритма. Согласно ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения» каждому действию алгоритма соответствует геометрическая фигура (блочный символ). Порядок выполнения действий указывается стрелками.

Перечень наиболее употребляемых символов приведен в таблице 1:

Таблица 1 – Наиболее часто употребляемые символы алгоритмов

Название символа	**Обозначение и пример заполнения**	**Пояснения**
Пуск-останов		Начало, завершение алгоритма или подпрограммы
Ввод-вывод данных		Ввод исходных данных или вывод результатов
Процесс, действие		Внутри прямоугольника записывается действие, например, расчетная формула
Условие, ветвление		Проверка условия, в зависимости от которого меняется направление выполнения алгоритма
Внешнее устройство		Запись/чтение информации на/из магнитного диска
Комментарий		Пояснения

Описание алгоритма с помощью блок схем осуществляется рисованием последовательности геометрических фигур, каждая из которых подразумевает выполнение определенного действия алгоритма.

Внутри каждого блока записывается соответствующее действие. Последовательность выполнения задается соединительной линией со стрелочкой. Последовательность выполнения сверху вниз и слева направо принята за основную.

Пример изображения алгоритма приготовления вареного картофеля в графическом виде приведен на рисунке 1.

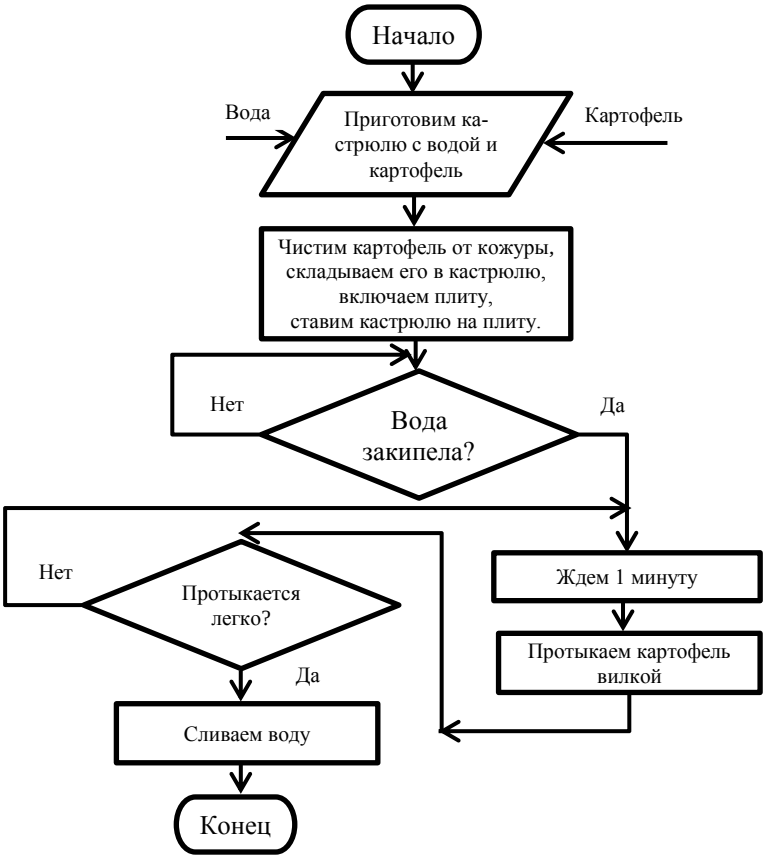


Рисунок 1 – Алгоритм приготовления вареного картофеля

Другим примером графического формата алгоритма можно указать пример с выкройками одежды. Графические изображения элементов выкройки сочетаются с текстовым описанием порядка соединения элементов.

Символьный вид алгоритма описывается с помощью набора символов или команд конкретного языка программирования, т.е. это текстовый формат программы.

В качестве основного вида записи мы будем пользоваться текстовым видом представления алгоритмов.

3.4 Задание на практическое занятие

1. Составить алгоритм формирования программной задержки в одну секунду. В качестве элемента, формирующего квант времени задержки, используем счетчик. Процесс уменьшения значения счетчик на единицу зани-

мает некоторое время. Это время будем считать квантом времени задержки. В нашем случае квант равен 1 микросекунде.

2. Для решения задачи необходимо использовать три регистра, работающие как один многоразрядный регистр.

3. Запишите алгоритм в текстовом виде и представьте его для проверки преподавателю.

3.5 Контрольные вопросы

1. Что такое алгоритм?
2. Как записываются алгоритмы?
3. Чем отличается текстовый вид записи алгоритма?
4. Что требуется для записи алгоритма в графическом виде?

4 ПРАКТИЧЕСКОЕ ЗАНЯТИЕ. РАЗРАБОТКА АЛГОРИТМА РАБОТЫ УСТРОЙСТВА

4.1 Цель занятия

Целью практического занятия является изучение процедуры разработки алгоритма функционирования реального устройства, реализуемого на основе микроконтроллера.

4.2 Языки программирования

Чтобы записать алгоритм в символьном виде, потребуется выбор языка программирования.

Язык программирования – это формализованный язык, предназначенный для описания программ и алгоритмов решения задач на ЭВМ. Языки программирования являются искусственными. В них синтаксис и семантика строго определены. Поэтому они не допускают свободного толкования выражения, что характерно для естественного языка. Языки программирования разделяются на две основные категории:

- языки высокого уровня;
- языки низкого уровня.

Язык высокого уровня - язык программирования, средства которого обеспечивают описание задачи в наглядном, легко воспринимаемом виде, удобном для программиста. Он не зависит от внутренних машинных кодов ЭВМ любого типа, поэтому программы, написанные на языках высокого уровня, требуют перевода в машинные коды программами транслятора либо интерпретатора. К языкам высокого уровня относят Фортран, ПЛ/1, Бейсик, Паскаль, Си, Ада и другие.

Язык низкого уровня - язык программирования, предназначенный для определенного типа ЭВМ и отражающий его внутренний машинный код. Язык низкого уровня называют еще как «машинный язык», «машинно-ориентированный язык» и «язык ассемблера».

4.3 Задание на практическое занятие

1. Составить алгоритм формирования реального устройства - светофора, работой которого управляет микроконтроллер.
2. Записать алгоритм функционирования светофора в текстовом виде.
3. Указать, какие изменения необходимо внести в алгоритм, чтобы реализовать устройство «Светофор, имеющий кнопку для пропуска пешехода».
4. Разработанные алгоритмы представьте для проверки преподавателю.

4.4 Контрольные вопросы

1. Язык программирования высокого уровня, что это?
2. Приведите примеры языков высокого уровня.
3. Язык программирования низкого уровня, что это?

5 ПРАКТИЧЕСКОЕ ЗАНЯТИЕ. РЕАЛИЗАЦИЯ АЛГОРИТМОВ В ВИДЕ ПРОГРАММ

5.1 Цель занятия

Целью практического занятия является изучение процедуры получения текста программы на ассемблере для микроконтроллера на основе разработанных алгоритмов.

5.2 Преобразование алгоритма в текст программы для микроконтроллера

Для преобразования алгоритма решения задачи в текст программы необходимо изучить набор команд конкретного микроконтроллера. В приложении А размещен набор всех команд микроконтроллера семейства MCS-51. Создадим программу на языке ассемблера.

При преобразовании алгоритма в текст программы следует учитывать, что некоторые строки алгоритма заменяются на команды микроконтроллера в соотношении «одна строка алгоритма - одна команда программы».

Другие строки алгоритма могут потребовать для реализации сразу по несколько команд микроконтроллера. Это значит, что при составлении алгоритма недостаточно была детализирована какая-то операция или шаг алгоритма.

Некоторые группы строк алгоритма вполне могут реализоваться с помощью одной команды микроконтроллера. В этом случае просто уровень детализации был несколько завышен.

Все указанные случаи являются вполне допустимыми и не свидетельствуют о недостатках конкретного алгоритма.

5.3 Пример реализации программы на языке ассемблера

В качестве примера реализации программы рассмотрим простую практическую задачу: Очистить, т.е обнулить заданное число ячеек резидентной памяти программ. Количество ячеек массива и начальный адрес очищаемого массива задается перед программой. Для практического решения задачи зададим начальный адрес очищаемого массива - 20. Длина массива -25 ячеек.

Составим алгоритм решения задачи. Программу можно реализовать записью 25 команд, каждая из которых записывает число 0 в конкретную ячейку памяти. Такой подход дает вполне работающую программу, но является неудобным. Размер программы практически линейно будет зависеть от количества ячеек очищаемого массива. И перестроить программу для другого диапазона адресов практически невозможно без полной переделки программы.

Очевидно, что программу следует организовать в виде повторяющегося цикла. На каждом этапе выполнения цикла будет очищаться одна ячейка.

Затем программа будет переходить к следующей ячейке. А для остановки цикла необходимо использовать счетчик, который будет указывать на то, сколько еще ячеек не обработано. Именно факт обнуления счетчика ячеек позволит завершить цикл и всю программу. В начале программы будут использоваться команды настройки начального адреса очищаемого массива и его длины. Такая реализация не изменяет размер программы при изменении адресов массива и его размера, что очень удобно.

Начитаем алгоритм с некоторой общей операции – ИНИЦИАЛИЗАЦИЯ. Это то место программы, к которому следует не забыть вернуться, когда будет разработано полное тело алгоритма. К этому моменту будут определены требуемые ресурсы для реализации алгоритма и нам необходимо будет задать их начальные значения.

1. Инициализация.
2. Записываем в ТЕКУЩУЮ ячейку памяти нуль (обнуляем элемент массива).
3. Настраиваемся на следующую ячейку памяти (следующий элемент массива.
4. Уменьшаем счетчик числа элементов.
5. Если счетчик не равен 0, то переходим на шаг 2, если равен 0, то идем на следующий шаг, т.е 6.
6. Останов программы.

Алгоритм вполне работоспособный, но требует использования некоторых ресурсов. Чтобы последовательно обрабатывать ячейки памяти, необходимо чтобы один регистр микроконтроллера хранил адрес текущей ячейки резидентной памяти данных, т.е. являлся указателем на ячейку памяти. Тогда простым увеличением значения в этом регистре мы будем готовы обратиться к следующей ячейке памяти.

Еще один регистр микроконтроллера будет выполнять функции счетчика числа элементов массива.

Для реализации программы ассемблера на языке микроконтроллера выберем регистры: указатель на массив – регистр R0, счетчик длины массива – регистр R7. Регистры умышленно выбраны не подряд, чтобы было понятно, что это не является важным. Важно, что выбраны конкретные регистры микроконтроллера.

Теперь понятно, что на шаге инициализации необходимо задать начальное значение указателю на начало массива R0=20 и счетчик числа ячеек массива R7=25.

Текст программы		**Алгоритм**
	MOV R0,#20 ; Указатель на массив	1. Инициализация.
	MOV R7, #25 ; Длина массива задана	
CIKL:	MOV @R0,#0;Чистим текущ.элемент	2. Записываем в

	ТЕКУЩУЮ ячейку памяти нуль (обнуляем элемент массива).
INC R0 ; Увеличиваем знач.указателя	3. Настраиваемся на следующую ячейку памяти (следующий элемент массива.
DJNZ R7,CIKL ; Организуем цикл	4. Уменьшаем счетчик числа элементов.
	5. Если счетчик не равен 0, то переходим на шаг 2, если равен 0, то идем на следующий шаг, т.е 6.
STOP: SJMP STOP ; Программная ловушка	6. Останов программы.

В тексте программы видно, что реализация алгоритма требует для некоторых шагов по несколько команд микроконтроллера (Шаг 1), некоторые шаги алгоритма реализуются одной командой на шаг алгоритма (Шаг 2, 3, 6). Имеется пример, когда несколько шагов алгоритма были реализованы с помощью одной команды (Шаг 4 и Шаг 5).

5.4 Задание на практическое занятие

1. Преобразовать алгоритм формирования задержки на 1 секунду, разработанный на предыдущих занятиях в программу на языке ассемблера.
2. Преобразовать алгоритм работы светофора в программу на языке ассемблера.
3. Преобразовать алгоритм работы «Светофор, имеющий кнопку для пропуска пешехода» в программу на языке ассемблера.
4. Разработанные программы на языке ассемблера представьте для проверки преподавателю.

5.5 Контрольные вопросы

1. Что необходимо выполнить при преобразовании алгоритма в программу на языке ассемблера?
2. Как соотносится число команд в программе на языке ассемблера с числом шагов алгоритма?
3. Почему иногда реализация программы в виде цикла предпочтительна линейной реализации программы?

6 САМОСТОЯТЕЛЬНАЯ РАБОТА ПО ДИСЦИПЛИНЕ

6.1 Состав дисциплины

Дисциплина «Вычислительная техника» рассчитана на один семестр и читается в четвертом семестре.

Объем дисциплины «Вычислительная техника» и виды учебной деятельности в 4-м семестре по учебному плану набора составляют (таблица 2):

Таблица 2 – Объем изучаемой дисциплины

Вид учебной деятельности	4 семестр	Единицы
Лекции	26	часов
Практические занятия	18	часов
Лабораторные работы	16	часов
Всего аудиторных занятий	60	часов
Самостоятельная работа	48	часов
Подготовка и сдача экзамена	36	часов
Общая трудоемкость по дисциплине	144	часа
Зачетные единицы	4	З.Е.

Формой промежуточной аттестации по учебному плану в 4 семестре по дисциплине является экзамен.

Объем самостоятельной работы составляет 48 часов.

6.2 Содержание самостоятельной подготовки

Самостоятельная работа включает: проработку материала лекций, подготовку к практическим занятиям, подготовку к контрольным работам, подготовку к выполнению лабораторных работ и написанию отчетов по ним, изучение тем теоретической части курса, отводимых на самостоятельную проработку.

Количество часов, отводимых на самостоятельную работу, и формы контроля самостоятельной работы сведены в таблицу 3.

Таблица 3 – Содержание самостоятельной работы по дисциплине

№	Наименование работы	Часы	Формы контроля
1.	Подготовка к лекциям	10 час	устный выборочный опрос по 5 мин перед лекцией
2.	Подготовка к практическим занятиям	8 час	устный выборочный опрос по 5 мин перед практикой
3.	Подготовка к лабораторным работам и оформление отчетов по ЛР	12 час	Допуск к лаборат. работам, защита отчетов по ЛР
4.	Изучение тем (вопросов) теоретической части курса, отводимых на само-	10 час	устный выборочный опрос по 5 мин перед лекцией

	стоятельную проработку: 1. Выполнение арифметических действий в модифицированном дополнительном коде. 2. Программные и аппаратные средства создания и отладки программного обеспечения. 3. Система команд микроконтроллера MCS-51.		
5.	Подготовка к контрольным работам: 1. Преобразование чисел между системами счисления. 2. Арифметические действия в разных системах счисления. 3. Микропроцессоры в системах управления. 4. ОМК MCS-51. 5. Периферийные устройства вычислительной техники	8 час	20 мин. контрольные работы перед практикой
	ИТОГО объем СРС	48 час	
	Подготовка и сдача экзамена	36 час	Оценка на экзамене

7 ПОДГОТОВКА К ЗАНЯТИЯМ

Подготовка к лекционным занятиям выполняется по материалу учебного пособия «Вычислительная техника. Учебное пособие».

Подготовка к выполнению лабораторных работ выполняется по материалу пособия «Вычислительная техника. Учебно-методическое пособие по организации лабораторных работ».

Подготовка к практическим занятиям выполняется по материалу данного пособия.

8 ТЕМЫ ДЛЯ САМОСТОЯТЕЛЬНОЙ ПРОРАБОТКИ

8.1 Выполнение арифметических действий в модифицированном дополнительном коде

На самостоятельную проработку выносится тема «Выполнение арифметических действий в модифицированном дополнительном коде», описанная в учебном пособии в разделе «Системы счисления в ЭВМ» в подразделе «Арифметические операции с числами».

Краткое изложение раздела.

Сложение в модифицированном коде.

Модифицированный код отличается наличием двух одинаковых знаковых битов. В модифицированном дополнительном коде у положительных чисел знаковые разряды равны нулю, а у отрицательных – единице. При выполнении операций правильным результатом будет число с одинаковыми

знаковыми битами. Наличие комбинации 01 или 10 в знаковых разрядов однозначно показывает переполнение разрядной сетки. А это убережет от использования ошибочного результата в дальнейших вычислениях. Рассмотрим сложение в модифицированном дополнительном коде. Сначала проверим предыдущий пример:

```
          [00]1100001
          [00]0111101
          [01]0011110
переносы        11   1
```

Мы видим, что результат содержит разные знаковые биты, поэтому он ошибочен. Ошибка объясняется тем, что для размещения значащей части результата необходимо 8, а не 7 разрядов, поскольку $N_1 = 97_{10}$, а $N_2 = 61_{10}$. Результат $N_1 + N_2 = 158_{10}$ больше предельного значения 127.

Рассмотрим сложение отрицательных чисел в модифицированном дополнительном коде. Числа уже преобразованы в дополнительный код и равны N_1 = [11]0110100 и N_2 = [11]0101101.

```
          [11]0110100
          [11]0101101
          [10]1100001
переносы   11  1111
```

В результирующем числе знаковые биты 10 сигнализируют о переполнении разрядной сетки, т.е. результат не помещается в семи двоичных разрядах. Действительно, если перевести числа в десятичную систему счисления, мы получим $N_1 = -76_{10}$, а $N_2 = -83_{10}$. Результат $N_1 + N_2 = -159_{10}$. Это действительно превышает значение -128_{10}, которое является наибольшим по модулю отрицательным числом, занимающим семь двоичных разрядов.

Рассмотрим другие примеры.

```
          [00]0110100
          [00]0101101
          [00]1100001
переносы        1111
```

Суммирование положительных чисел дало правильный результат. Действительно, $N_1 = 52_{10}$, а $N_2 = 45_{10}$. Результат $N_1 + N_2 = 97_{10}$. Рассмотрим еще один пример.

```
          [11]1100001
          [11]0111101
          [11]0011110
переносы   11 11      1
```

Результат правильный и в десятичных числах записывается как $N_1 = -31_{10}$, а $N_2 = -67_{10}$. Результат $N_1 + N_2 = -98_{10}$. Для следующего примера возьмем числа с разными знаками $N_1 = -52_{10}$, а $N_2 = +45_{10}$. Результат $N_1 + N_2 = -7_{10}$.

```
          [11]1001100
          [00]0101101
          [11]1111001
переносы        11
```

Результат получился правильным и представлен в модифицированном дополнительном коде. Можете поверить на слово, что и сложение большого по модулю положительного числа с маленьким (по модулю) отрицательным числом даст правильный положительный результат.

8.2 Программные и аппаратные средства создания и отладки программного обеспечения

На самостоятельную проработку выносится тема «Программные и аппаратные средства создания и отладки программного обеспечения».

Краткое изложение раздела.

Кросс-технология разработки программного обеспечения. Микро-ЭВМ и микроконтроллеры (МК) для радиоэлектронных устройств и систем предназначены для решения задач управления и обработки сигналов. От обычных микро-ЭВМ они отличаются способом разработки программного обеспечения. Микро-ЭВМ и МК, встроенные в оборудование, не имеют соответствующего набора необходимых внешних устройств (дисплеи, принтеры, внешние накопители и т.д.) и поэтому не пригодны для разработки и отладки программного обеспечения. Часто в этом случае программы разрабатываются и отлаживаются на универсальных ЭВМ, с использованием пакета специальных программ. При этом приходится разрабатывать программы на языке ассемблера. Отладку выполняют с помощью симулятора – программной модели микро-ЭВМ или контроллера.

Этот способ разработки программного обеспечения называется кросс-технологией, в противовес резидентной технологии. При разработке программ на языке ассемблера используют различные служебные программы – редакторы текста, трансляторы, компоновщики, отладчики программ.

В кросс-технологии применяются: кросс-транслятор, кросс-компоновщик, кросс-отладчик. В минимальный пакет кросс-программ разработки программного обеспечения (ПО) для управляющей микро-ЭВМ входят:

- ***редактор текста***, обеспечивающий ввод программы на языке ассемблера микро-ЭВМ или МК, редактирование и запись файла программы;

- ***кросс-транслятор***, преобразующий исходный текст программы в файл объектного кода, содержащего коды команд управляющей микро-ЭВМ (МК), информацию для редактора связей, предупреждения и сообщения об ошибках;
- ***кросс-компоновщик объектных модулей (кросс-редактор связей)***. Компоновщик связывает несколько объектных модулей в единую программу, добавляет вызываемые из программы библиотечные файлы, подставляет вместо ссылок на объекты их реальные или относительные адреса и формирует исполняемый файл в кодах микро-ЭВМ или МК;
- ***кросс-отладчик (симулятор)***, описывающий программную логическую модель микро-ЭВМ или микроконтроллера. При работе симулятор дает эффективный и удобный механизм покомандной и модульной отладки компонентов программы с доступом к программно эмулируемым ресурсам микро-ЭВМ: регистрам, ячейкам памяти, таймерам, портам и т.д.

Перед составлением текста программы необходимо разработать алгоритм решения поставленной задачи в виде последовательного набора шагов (элементарных действий), позволяющий решить требуемую задачу.

Каждую строку алгоритма распишем в виде команд – текст программы готов. С этого момента нужны специальные программные средства.

Общая процедура создания и отладки программ для микро-ЭВМ или МК включает этапы, показанные на рисунке 2.

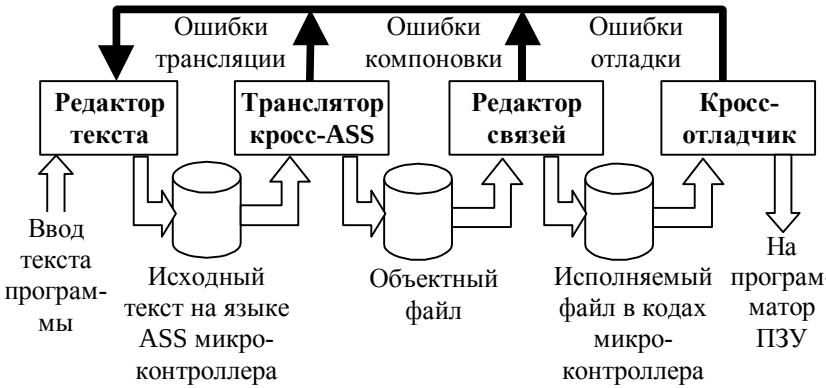


Рисунок 2 – Алгоритм создания программного обеспечения для микроконтроллера

Исходный текст программы с помощью редактора текста вводится в память универсальной ЭВМ и записывается в файл. Затем с помощью кросс-транслятора этот файл переводится в промежуточную форму – объектный файл. Объектный файл требует дополнительной обработки, которая выполняется с помощью программы – кросс-редактора связей. На выходе редакто-

ра связей формируется двоичный файл, который может быть выполнен на реальной микро-ЭВМ или МК. Для проверки функционирования созданной программы можно выполнить ее отладку в симуляторе. Отлаженный файл программы в кодах микро-ЭВМ (МК) записывается с помощью программатора в постоянную память (ПЗУ) микропроцессорной системы.

В простейших случаях операции трансляции и компоновки могут выполняться совместно без записи объектного файла на диск. При этом транслятор и компоновщик объединяются в одной программе, которая выполняет несколько последовательных проходов (просмотров) файла, и преобразует исходный текст программы в исполняемый код.

На каждом из этапов создания программного обеспечения могут быть обнаружены различные ошибки. Это приводит к необходимости возврата к исходному тексту программы, исправления его и повторения всей процедуры преобразования.

Ошибки трансляции связаны с неправильными типами операндов команд, пропуском необходимых элементов оформления текста программы, опечатками, синтаксическими ошибками записи команд и другими ошибками, обнаруживаемыми по внешнему виду команды. Эти ошибки являются самыми простыми и легко устраняются при внимательном прочтении указанной транслятором ошибочной строки.

Ошибки компоновки проявляются на этапе работы редактора связей и означают неправильные указания для объединения нескольких модулей подпрограмм в единую общую программу. Здесь встречаются ссылки на неизвестные программы и метки, ошибки в именах программных модулей, неправильное использование библиотечных функций. Осознание и устранение подобных ошибок, как правило, требует простой внимательности и не вызывает больших трудностей.

Нередко компоновщики при обнаружении ошибок, которые на их взгляд не являются критическими, выдают просто предупреждения. При этом они создают исполняемый файл. В таком файле ошибочные ссылки заменяются стандартными значениями, например, нулями. Опасность скрывается в игнорировании предупреждений компоновщика и использовании исполняемого файла с ошибками связей. В этом случае результат работы программы непредсказуем.

Ошибки исполняемого файла связаны с ошибками алгоритма, неправильной работой программы, зацикливаниями, ошибками счета, ошибками преобразования и другими ошибками. Такие ошибки проявляются и обнаруживаются только при исполнении программы. Как правило, это самые трудные для диагностирования и устранения ошибки. Анализ сложной и громоздкой управляющей или обрабатывающей программы требует внимательности, памяти, хороших знаний поведения объекта управления и занимает продолжительное время.

Ситуация несколько упрощается при использовании программных моделей – симуляторов. Перед записью программы в ПЗУ, ошибки данного

этапа можно обнаружить, исследовать и найти пути устранения с помощью программы кросс-отладчика. Программная модель микро-ЭВМ позволяет проводить анализ исполняемого кода с использованием большого числа сервисных возможностей, которые отсутствуют в реально работающей аппаратной ЭВМ.

Перечисленные выше возможности обеспечивают на разных этапах создания и преобразования программ выявление и устранение большинства ошибок. Однако нельзя гарантировать устранение всех ошибок на 100%. Кроме перечисленных, остаются еще динамические ошибки, проявляющие себя лишь при реальной работе аппаратной микро-ЭВМ. Эти ошибки могут быть связаны с реальными режимами прерываний, условиями функционирования портов, таймеров, ячеек памяти. По этой причине достаточно сложные программные модули даже после тщательной отладки могут еще содержать не выявленные ошибки и обычно их содержат.

8.3 Система команд микроконтроллера MCS-51

На самостоятельную проработку выносится тема «Система команд микроконтроллера MCS-51», описанная в учебном пособии в разделе «Однокристальный микроконтроллер семейства MCS-51» в подразделе «Система команд ОЭВМ КР1816ВЕ51».

Краткое изложение раздела.

Система команд MCS-51 содержит 111 базовых команд. По функциональному признаку команды можно разделить на пять групп:

a) команды передачи данных;
b) арифметические команды;
c) логические команды;
d) команды передачи управления;
e) команды операций с битами.

Большинство команд (94) имеют формат один или два байта и выполняются за один или два машинных цикла. Остальные команды – трехбайтные и выполняются за два машинных цикла. Только на выполнение команд умножения и деления требуется четыре машинных цикла.

Первый байт всех 13 типов команд содержит код операции (КОП). Второй и третий байты содержат либо адреса операндов, либо непосредственные данные.

В системе команд MCS-51 используется четыре способа адресации данных.

1. Регистровая адресация. В команде указано символическое название регистра.
2. Прямая адресация. В команде явно указан адрес операнда в памяти.
3. Непосредственная адресация. В команде задан сам операнд.

4. Косвенная адресация. В команде задано имя регистра или регистровой пары, где содержится адрес, по которому и происходит обращение к операнду в памяти.

В системе команд используются следующие обозначения (в скобках показаны варианты обозначений, встречающиеся в командах, с буквой h обозначаются старшие байты слов, с буквой l – младшие байты слов):

Rn, n=0÷7 – регистры R0-R7 текущего банка регистров;

direct (ad, add, ads) – 8-битный прямой адрес ячейки резидентной памяти данных (0-127) или регистра специальных функций (128-255);

@Ri, i=0,1 – 8-битная ячейка РПД или регистра специальных функций, косвенно адресуемая через регистр R0 или R1;

#data (#d) – 8-битная константа, входящая в состав команды;

#data16 (#d16h, #d16l) – 16-битная константа, входящая в состав команды (непосредственная адресация);

addr16 (ad16h, ad16l) – 16-битовый адрес назначения в командах длинного перехода LJMP и вызова подпрограмм LCALL, позволяющий охватить полное адресное пространство в 64 Кбайт;

addr11 – 11-битовый адрес назначения в командах абсолютного перехода AJMP и вызова подпрограмм ACALL, позволяющий выполнить переход в пределах 2 Кбайт страницы от адреса текущей команды;

rel – 8-битное смещение со знаком. Смещение с учетом знака суммируется с текущим содержимым счетчика команд и позволяет выполнить переход в пределах –128 +127 байт относительно текущего адреса;

bit – 8-битный прямой адрес бита, находящегося в РПД (0-127) или в регистре специальных функций (128-255).

Микроконтроллер оперирует данными четырех типов: биты, тетрады (4-битные цифры), байты и 16-битные слова.

Биты. В MCS-51 определены 256 битов, из них 128 находятся в РПД и используются как флаги пользователя, а остальные 128 битов расположены в блоке регистров специальных функций и входят в состав некоторых регистров этого блока. Для обращения к битам используется прямой 8-битный адрес (bit).

Тетрады. Операции с тетрадами определены только в командах обмена тетрад в аккумуляторе и между регистром и аккумулятором.

Байты. Байтовым операндом может быть адрес внутренней и внешней памяти данных и программ, непосредственный операнд, константа, адрес регистра специальных функций или порта ввода/вывода.

Слова. Двухбайтовые операнды могут быть или константами, или прямыми адресами памяти и занимают второй и третий байты команды.

В слове состояния процессора PSW имеются четыре флага, отражающие результат операции в АЛУ: C – перенос, AC – вспомогательный перенос, OV – переполнение и P – паритет. Только некоторые команды изменяют флаги результата.

Команды передачи данных

Команды передачи данных не влияют на флаги результата.

Большую часть команд данной группы составляют команды передачи и обмена байтами.

На графе возможных путей передачи данных в МК (рисунок 3 **33**) аккумулятор представлен отдельной вершиной, поскольку многие команды используют его при передаче данных.

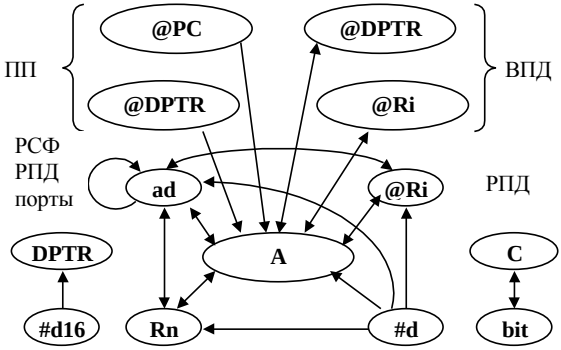


Рисунок 3 – Граф путей передачи данных

Полный перечень команд передачи данных представлен в приложении А «Список команд ОЭВМ MCS-51» в подразделе «Группа команд передачи данных».

Арифметические команды

В данную группу команд входят команды сложения, вычитания, умножения и деления байтов. Имеются команды десятичной коррекции результата сложения, увеличения на единицу как 8-битовых, так и 16-битовых операндов, уменьшения на единицу для 8-битовых операндов.

Все команды этой группы используют АЛУ и влияют на состояние флагов.

Полный перечень арифметических команд представлен в приложении А «Список команд ОЭВМ MCS-51» в подразделе «Группа арифметических операций».

Логические команды

В данную группу входят команды, выполняющие логические операции вида «И», «ИЛИ», «Исключающее ИЛИ» над операндом в аккумуляторе и в любом байте РПД или в регистрах специальных функций. Сюда же входят команды сдвига аккумулятора влево и вправо, а также сброса и инверсии аккумулятора.

Все команды этой группы используют АЛУ и влияют на состояние флагов.

Полный перечень логических команд представлен в приложении А «Список команд ОЭВМ MCS-51» в подразделе «Группа логических операций».

Команды операций с битами

Данные команды оперируют с однобитовыми операндами. Биты расположены в области битовой адресации (128 битов пользователя) и в области регистров специальных функций. Для адресации бит используется прямой 8-битовый адрес (bit). Косвенная адресация бит невозможна.

Команды не затрагивают флагов, кроме флага переноса в отдельных случаях.

Полный перечень команд операций с битами представлен в приложении А «Список команд ОЭВМ MCS-51» в подразделе «Группа битовых операций».

Команды передачи управления

Данная группа команд содержит команды условных и безусловных переходов, вызовы подпрограмм и возврат из них, команды сравнения и цикла. Для выполнения условных переходов можно проверять разнообразные условия, а безусловные переходы и вызовы подпрограмм различаются по предельной удаленности перехода

Полный перечень команд передачи управления представлен в приложении А «Список команд ОЭВМ MCS-51» в подразделе «Группа команд передачи управления».

9 ПОДГОТОВКА К КОНТРОЛЬНЫМ РАБОТАМ

Для подготовки к контрольным работам следует проработать следующие разделы лекционных разделов учебного пособия.

Для контрольной работы «Преобразование чисел между системами счисления» следует проработать раздел 2.2 учебного пособия.

Для контрольной работы «Арифметические действия в разных системах счисления» следует проработать раздел 2.3 учебного пособия.

Для контрольной работы «Микропроцессоры в системах управления» следует проработать раздел 3 учебного пособия.

Для контрольной работы «ОМК MCS-51» следует проработать раздел 4 учебного пособия.

Для контрольной работы «Периферийные устройства вычислительной техники» следует проработать раздел 5 учебного пособия.

ПРИЛОЖЕНИЕ А

(справочное)

СПИСОК КОМАНД ОЭВМ MCS-51

Мнемокод	Название	байт	цикл	Операция
Группа команд передачи данных				
MOV @Ri,#d	Пересылка в РПД константы (i=0,1)	2	1	**((Ri))←#d**
MOV @Ri,A	Пересылка в РПД байта из аккумулятора	1	1	**((Ri))←(A)**
MOV @Ri,ad	Пересылка в РПД байта с прям.адреса ad	2	2	**((Ri))←(ad)**
MOV A,#d	Загрузка в аккумулятор константы	2	1	**(A)←#d**
MOV A,@Ri	Пересылка в аккумулятор из РПД (i=0,1)	1	1	**(A)←((Ri))**
MOV A,ad	Пересылка в аккумулятор байта с адреса ad	2	1	**(A)←(ad)**
MOV A,Rn	Пересылка в аккумулятор байта из регистра (n=0÷7)	1	1	**(A)←(Rn)**
MOV ad,#d	Пересылка константы по прямом.адресу	3	2	**(ad)←#d**
MOV ad,@Ri	Пересылка по прямому адресу байта из РПД (i=0,1)	2	2	**(ad)←((Ri))**
MOV ad,A	Пересылка аккумулятора по прямому адресу	2	1	**(ad)←(A)**
MOV ad,Rn	Пересылка регистра по прямому адресу	2	2	**(ad)←(Rn)**
MOV add,ads	Пересылка прямоадресуемого байта по прямому адресу	3	2	**(add)←(ads)**
MOV DPTR,#d16	Загрузка указателя данных словом	3	2	**(DPTR)←#d16**
MOV Rn,#d	Загрузка в регистр (n=0÷7) константы	2	1	**(Rn)←#d**
MOV Rn,A	Пересылка в регистр (n=0÷7) байта из аккумулятора	1	1	**(Rn)←(A)**
MOV Rn,ad	Пересылка в регистр с прямого адреса	2	2	**(Rn)←(ad)**
MOVC A,@A+DPTR	Пересылка в аккумулятор байта из ПП	1	2	**(A)←((A)+(DPTR))**

Мнемокод	Название	байт	цикл	Операция
MOVC A,@A+PC	Пересылка в аккумулятор байта из ПП	1	2	**(A)←((A)+(PC))**
MOVX @DPTR,A	Пересылка в расшир. ВПД из аккумулятора	1	2	**((DPTR))←(A)**
MOVX A,@DPTR	Пересылка в аккумулятор из расшир. ВПД	1	2	**(A)←((DPTR))**
MOVX @Ri,A	Пересылка в ВПД из аккумулятора	1	2	**((Ri))←(A)**
MOVX A,@Ri	Пересылка в аккумулятор байта из ВПД	1	2	**(A)←((Ri))**
POP ad	Извлечь из стека	2	2	**(ad)←((SP))** **(SP)←(SP)-1**
PUSH ad	Загрузить в стек	2	2	**(SP)←(SP)+1** **((SP))←(ad)**
XCH A,@Ri	Обмен аккумулятора с байтом из РПД	1	1	**(A)↔((Ri))**
XCH A,ad	Обмен аккумулятора и байта с адреса ad	2	1	**(A)↔(ad)**
XCH A,Rn	Обмен аккумулятора с регистром	1	1	**(A)↔(Rn)**
XCHD A,@Ri	Обменять младш. тетраду аккумулятора с мл. тетрадой байта РПД	1	1	**$(A_{0-3})\leftrightarrow((Ri)_{0-3})$**
SWAP A	Обменять тетрады в аккумуляторе	1	1	**$(A_{0-3})\leftrightarrow(A_{4-7})$**
Группа арифметических операций				
ADD A,#d	Сложить аккумулятор с константой	2	1	**(A)←(A)+#d**
ADD A,@Ri	Сложить аккумулятор с байтом из РПД	1	1	**(A)←(A)+((Ri))**
ADD A,ad	Сложить аккумулятор с байтом по адресу ad	2	1	**(A)←(A)+(ad)**
ADD A,Rn	Сложить аккумулятор с регистром	1	1	**(A)←(A)+(Rn)**
ADDC A,#d	Сложить аккумулятор с константой и переносом	2	1	**(A)←(A)+#d+(C)**
ADDC A,@Ri	Сложить аккумулятор с байтом из РПД (i=0,1) и переносом	1	1	**(A)←(A)+((Ri))+(C)**
ADDC A,ad	Сложить аккумулятор с байтом по адресу ad и переносом	2	1	**(A)←(A)+(ad)+(C)**
ADDC A,Rn	Сложить аккумулятор с регистром (n=0÷7) и переносом	1	1	**(A)←(A)+(Rn)+(C)**

Мнемокод	Название	байт	цикл	Операция
DA A	Десятичная коррекция аккумулятора	1	1	
SUBB A,#d	Вычесть из аккумулятора константу и заем	2	1	**(A)←(A)-#d-(C)**
SUBB A,@Ri	Вычесть из аккумулятора байт из РПД и заем	1	1	**(A)←(A)-((Ri))-(C)**
SUBB A,ad	Вычесть из аккумулятора байт с адреса ad и заем	2	1	**(A)←(A)-(ad)-(C)**
SUBB A,Rn	Вычесть из аккумулятора регистр и заем	1	1	**(A)←(A)-(Rn)-(C)**
INC @Ri	Инкремент байта в РПД	1	1	**((Ri))←((Ri))+1**
INC A	Инкремент аккумулятора	1	1	**(A)←(A)+1**
INC ad	Инкремент байта по адр. ad	2	1	**(ad)←(ad)+1**
INC DPTR	Инкремент указателя данных	1	2	**(DPTR)←(DPTR)+1**
INC Rn	Инкремент регистра	1	1	**(Rn)←(Rn)+1**
DEC @Ri	Декремент байта в РПД	1	1	**((Ri))←((Ri))-1**
DEC A	Декремент аккумулятора	1	1	**(A)←(A)-1**
DEC ad	Декремент байта по адресу ad	2	1	**(ad)←(ad)-1**
DEC Rn	Декремент регистра	1	1	**(Rn)←(Rn)-1**
MUL AB	Умножение аккумулятора на регистр B	1	4	**(B)(A)←(A)·(B)**
DIV AB	Деление аккумулятора на регистр B	1	4	**(A).(B)←(A)/(B)**
Группа логических операций				
ANL A,#d	Логическое И аккумулятора и константы	2	1	**(A)←(A)∧#d**
ANL A,@Ri	Логическое И аккумулятора с байтом из РПД (i=0,1)	1	1	**(A)←(A)∧((Ri))**
ANL A,ad	Логическое И аккумулятора с байтом по адресу ad	2	1	**(A)←(A)∧(ad)**
ANL A,Rn	Логическое И аккумулятора с регистром (n=0÷7)	1	1	**(A)←(A)∧(Rn)**
ANL ad,#d	Логическое И байта по адресу ad с константой	3	2	**(ad)←(ad)∧#d**
ANL ad,A	Логическое И байта по адресу ad с аккумулятором	2	1	**(ad)←(ad)∧(A)**
ORL A,#d	Логическое ИЛИ аккумулятора с константой	2	1	**(A)←(A)∨#d**

Мнемокод	Название	байт	цикл	Операция
ORL A,@Ri	Логическое ИЛИ аккумулятора с байтом из РПД (i=0,1)	1	1	**(A)←(A)∨((Ri))**
ORL A,ad	Логическое ИЛИ аккумулятора с байтом по адресу ad	2	1	**(A)←(A)∨(ad)**
ORL A,Rn	Логическое ИЛИ аккумулятора с регистром (n=0÷7)	1	1	**(A)←(A)∨(Rn)**
ORL ad,#d	Логическое ИЛИ байта по адресу ad с константой	3	2	**(ad)←(ad)∨#d**
ORL ad,A	Логическое ИЛИ байта по адресу ad с аккумулятором	2	1	**(ad)←(ad)∨(A)**
XRL A,#d	Исключ. ИЛИ аккумулятора с константой	2	1	**(A)←(A)⊕#d**
XRL A,@Ri	Исключающее ИЛИ аккумулятора с байтом из РПД (i=0,1)	**1**	**1**	**(A)←(A)⊕((Ri))**
XRL A,ad	Исключающее ИЛИ аккумулятора с байтом по адресу ad	2	1	**(A)←(A)⊕(ad)**
XRL A,Rn	Исключающее ИЛИ аккумулятора с регистром (n=0÷7)	1	1	**(A)←(A)⊕(Rn)**
XRL ad,#d	Исключающее ИЛИ байта по адресу ad с константой	3	2	**(ad)←(ad)⊕#d**
XRL ad,A	Исключающее ИЛИ байта по адресу ad с аккумулятором	2	1	**(ad)←(ad)⊕(A)**
CLR A	Сброс аккумулятора	1	1	**(A)←0**
CPL A	Инверсия аккумулятора	1	1	**(A)←($\overline{A}$)**
RL A	Сдвиг аккумулятора влево по циклу	1	1	**(A_{n+1})←(A_n), n=0÷6 (A_0)←(A_7)**
RR A	Сдвиг аккумулятора вправо по циклу	1	1	**(A_n)←(A_{n+1}), n=0÷6 (A_7)←(A_0)**
RLC A	Сдвиг аккумулятора влево через перенос	1	1	**(A_{n+1})←(A_n), n=0÷6 (A_0)←(C) (C)←(A7)**
RRC A	Сдвиг аккумулятора вправо через перенос	1	1	**(A_n)←(A_{n+1}), n=0÷6 (A_7)←(C) (C)←(A_0)**
Группа битовых операций				
MOV bit,C	Пересылка разряда переноса в бит	2	2	**(b)←(C)**
MOV C,bit	Пересылка бита в разряд переноса	2	1	**(C)←(b)**
ANL C,bit	Логическое И бита и переноса	2	2	**(C)←(C)∧(b)**
ANL C,/bit	Логическое И инверсии бита и переноса	2	2	**(C)←(C)∧($\overline{b}$)**

Мнемокод	Название	байт	цикл	Операция
ORL C,bit	Логическое ИЛИ бита и переноса	2	2	**(C)←(C)∨(b)**
ORL C,/bit	Логическое ИЛИ инверсии бита и переноса	2	2	**(C)←(C)∨($\overline{b}$)**
CLR bit	Сброс бита	2	1	**(b)←0**
CLR C	Сброс переноса	1	1	**(C)←0**
SETB bit	Установка бита	2	1	**(b)←1**
SETB C	Установка переноса	1	1	**(C)←1**
CPL bit	Инверсия бита	2	1	**(b)←($\overline{b}$)**
CPL C	Инверсия переноса	1	1	**(C)←($\overline{C}$)**
Группа команд передачи управления				
LCALL ad16	Длинный вызов программы	3	2	**(PC)←(PC)+3, (SP)←(SP)+2, (SP-1)←(PC_{0-7}), (SP)←(PC_{8-15}), (PC)←ad16**
ACALL ad11	Абсолютный вызов подпрограммы в странице 2 Кбайта	2	2	**(PC)←(PC)+2, (SP)←(SP)+2, (SP-1)←(PC_{0-7}), (SP)←(PC_{8-15}), (PC_{0-10})←ad11**
LJMP ad16	Длинный переход	3	2	**(PC)←ad16**
AJMP ad11	Абсолютный переход в странице размером 2 Кбайт	2	2	**(PC_{0-10})←ad11**
SJMP rel	Короткий относительный переход	2	2	**(PC)←(PC)+2+rel**
CJNE A,ad,rel	Сравнение аккумулятора с байтом по адресу ad и переход, если не равно	3	2	**(PC)←(PC)+3, если (A)≠(ad), то (PC)←(PC)+rel**
CJNE @Ri,#d,rel	Сравнение байта из РПД с константой и переход, если не равно	3	2	**(PC)←(PC)+3, если ((Ri))≠#d, то (PC)←(PC)+rel**
CJNE A,#d,rel	Сравнение аккумулятора с константой и переход, если не равно	3	2	**(PC)←(PC)+3, если (A)≠#d, то (PC)←(PC)+rel**
CJNE Rn,#d,rel	Сравнение регистра с константой и переход, если не равно	3	2	**(PC)←(PC)+3, если (Rn)≠#d, то (PC)←(PC)+rel**

Мнемокод	Название	Байт	Цикл	Операция
DJNZ ad,rel	Декремент байта по адресу ad и переход, если не нуль	3	2	**(PC)←(PC)+2, (ad)←(ad)-1, если (ad)≠0, то (PC)←(PC)+rel**
DJNZ Rn,rel	Декремент регистра и переход, если не нуль	2	2	**(PC)←(PC)+2, (Rn)←(Rn)-1, если (Rn)≠0, то (PC)←(PC)+rel**
JB bit,rel	Переход, если бит равен единице	3	2	**(PC)←(PC)+3, если (b)=1, то (PC)←(PC)+rel**
JBC bit,rel	Переход, если бит установлен, с последующим сбросом бита	3	2	**(PC)←(PC)+3, если (b)=1, то (b)←0, (PC)←(PC)+rel**
JC rel	Переход, если перенос равен единице	2	2	**(PC)←(PC)+2, если (C)=1, то (PC)←(PC)+rel**
JMP @A+DPTR	Косвенный относительный переход	1	2	**(PC)←(A)+(DPTR)**
JNB bit,rel	Переход, если бит равен нулю	3	2	**(PC)←(PC)+3, если (b)=0, то (PC)←(PC)+rel**
JNC rel	Переход, если перенос равен нулю	2	2	**(PC)←(PC)+2, если (C)=0, то (PC)←(PC)+rel**
JNZ rel	Переход, если аккумулятор не равен нулю	2	2	**(PC)←(PC)+2, если (A)≠0, то (PC)←(PC)+rel**
JZ rel	Переход, если аккумулятор равен нулю	2	2	**(PC)←(PC)+2, если (A)=0, то (PC)←(PC)+rel**
RET	Возврат из подпрограммы	1	2	**(PC_{8-15})←((SP)) (PC_{0-7})←((SP)-1) (SP)←(SP)-2**
RETI	Возврат из подпрограммы обработки прерывания	1	2	**(PC_{8-15})←((SP)) (PC_{0-7})←((SP)-1) (SP)←(SP)-2**
NOP	Нет операции	1	1	**(PC)←(PC)+1**