ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Кафедра «Вычислительные системы и информационная безопасность»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к проведению лабораторных работ
по дисциплине

«Аппаратные средства вычислительной техники»

Авторы
Айдинян А.Р.
Маршаков Д.В.

Ростов-на-Дону, 2014

Аннотация

В компактной форме приводятся краткие теоретические сведения к выполнению лабораторных работ, порядок выполнения лабораторных работ, индивидуальные задания и контрольные вопросы.

Автор

к.т.н., доцент
Айдинян А.Р.

к.т.н., доцент
Маршаков Д.В.

Оглавление

ЛАБОРАТОРНАЯ РАБОТА 1
ПРЕДСТАВЛЕНИЕ ДАННЫХ В ЭВМ И МАШИННАЯ АРИФМЕТИКА

Цель работы: Изучить способы кодирования целых и вещественных чисел и способы выполнения арифметических операций в двоичном коде.

Краткие теоретические сведения

Для представления информации в памяти ЭВМ (как числовой, так и не числовой) используется двоичный способ кодирования.

Элементарная ячейка памяти ЭВМ имеет длину 8 бит (байт). Наибольшую последовательность бит, которую ЭВМ может обрабатывать как единое целое, называют *машинным словом*. Длина машинного слова зависит от разрядности процессора и может быть равной 16, 32, 64 битам и т.д.

Разряды нумеруются справа налево, начиная с 0. На рис. 1 показана нумерация бит в двухбайтовом машинном слове.

№ бита	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Двоичный код	…	…	…	…	…	…	0	0	0	0	0	1	1	1	1	1

Рис. 1. Нумерация бит в двухбайтовом машинном слове

Для записи двоичного числа кроме символов '0' и '1' применяются символы '+', '–', двоичная запятая.

В двоичном коде набор символов ограничен нулем и единицей и занимает определенное количество разрядов, поэтому:

- для записи символов '+' и '–' в двоичном коде используется 0 и 1 и отводится определенный разряд.
- двоичная запятая не ставится, а подразумевается между определенными разрядами двоичного кода.
- незначащие разряды двоичного числа в двоичном коде заполняются нулями, либо повторением знака числа в зависимости от вида двоичного кода.

Виды двоичных кодов.

Целые беззнаковые коды.

Диапазон значений величин зависит от количества бит памяти, отведенных для их хранения. Для беззнакового

представления целых двоичных числа изменяются от 0 до $2^n - 1$, а в представлении со знаком — от -2^{n-1} до $2^{n-1} - 1$.

Для определённости примем длину слова процессора равной восьми битам. В этих кодах каждый двоичный разряд представляет собой степень цифры 2:

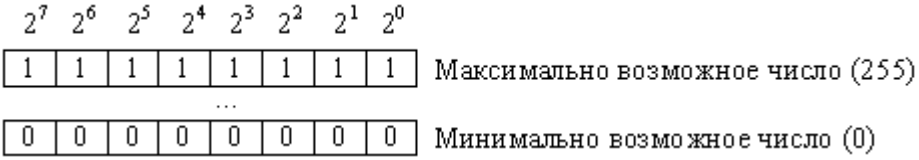


Рисунок 2. Целые беззнаковые коды

На рисунке 2 над каждым разрядом беззнакового двоичного кода приведено значение его веса. При этом минимально возможное число, которое можно записать таким двоичным кодом, равно 0. Максимально возможное число, которое можно представить этим кодом, определяется следующей формулой:

$$M = 2^n - 1$$

где n – разрядность двоичного числа. Разрядность числа обычно выбирают кратной разрядности микропроцессора.

Эти два числа полностью определяют диапазон значений чисел, которые можно представить двоичным кодом. В случае двоичного 8-разрядного беззнакового двоичного кода целые числа, которые можно записать с его помощью, находятся в диапазоне от 0 до 255. Восьмиразрядное двоичное число обычно называют байтом иногда его еще называют октетом или полусловом.

Для беззнакового двоичного 16-разрядного кода диапазон представляемых значений будет от 0 до 65535. В микропроцессорной системе, построенной на 8-разрядном процессоре, для хранения 16-разрядного числа используются две ячейки памяти, расположенные в соседних адресах.

В качестве примера записи 16-разрядного числа в двух соседних 8 разрядных ячейках памяти, на рисунке 3 приведена запись числа $56249_{10} = 1101101110111001_2$.

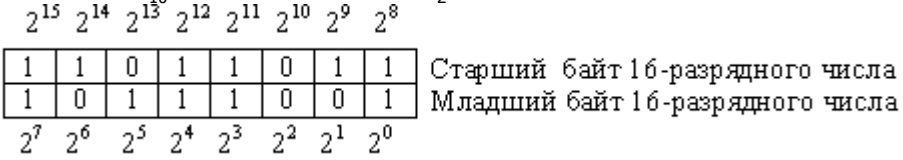


Рисунок 3. Формат 16-разрядного беззнакового двоичного кода, записанного в двух соседних ячейках памяти

Следует отметить, что в отличие от действий в математике, при записи числа в прямом двоичном коде не существует возможности не записывать старшие незначащие нули. Поэтому число 12_{10} не может быть записано как 1100. Оно обязательно должно быть записано во всех восьми разрядах кода: 00001100. Если число первоначально было определено как 16-разрядное, то в этом случае это же самое число 1210 должно быть записано как 0000000000001100. В качестве еще одного примера рассмотрим запись числа 31_{10}. Оно будет записано в 8-разрядном прямом беззнаковом двоичном коде как 00011111, а в 16 разрядном - 0000000000011111.

Прямые целые знаковые двоичные коды. В этих кодах старший разряд в слове используется для представления знака числа.

В прямом знаковом коде нулем обозначается знак '+', а единицей — знак '-'. В случае представления величины со знаком самый левый (старший) разряд указывает на положительное число, если содержит нуль, и на отрицательное, если – единицу.

В результате введения знакового разряда диапазон чисел смещается в сторону отрицательных чисел. Формат 8-разрядного прямого знакового двоичного кода приведен на рисунке 4. На рисунке приведено шесть различных чисел, записанных в этом коде. Слева указаны десятичные числа, соответствующие двоичному представлению обратного кода, а сверху, над каждым разрядом двоичного кода, указан его вес.

Рисунок 4. Прямые целые знаковые коды

В случае двоичного восьмиразрядного знакового целого числа диапазон чисел, которые можно записать таким кодом: -127 .. +127. Для шестнадцатиразрядного кода этот диапазон будет: -32767 .. +32767. В восьмиразрядном процессоре для хранения такого числа тоже используется две ячейки памяти, расположенные в соседних адресах.

Обратите внимание, что при считывании содержимого памяти микропроцессора мы ни по каким внешним признакам не сможем отличить знаковый двоичный код от беззнакового. Это сможет сделать только программа, в которой заложена информация о том, в каких ячейках памяти какой код следует использовать для хранения чисел. В результате человек или программа, не обладающие подобной информацией, при попытке прочитать ячейки памяти не смогут узнать – какое же число записано в данных конкретных ячейках памяти.

Недостатком такого кода является то, что знаковый разряд и цифровые разряды приходится обрабатывать раздельно. Алгоритм программ, работающий с такими кодами получается сложный. Для выделения и изменения знакового разряда приходится применять механизм маскирования разрядов, что резко увеличивает размер программы и уменьшает ее быстродействие. Для того, чтобы алгоритм обработки знакового и цифровых разрядов не различался, были введены обратные двоичные коды.

Обратные двоичные коды. Обратные двоичные коды отличаются от прямых только тем, что отрицательные числа в них

получаются инвертированием всех разрядов числа. При этом знаковый и цифровые разряды не различаются. Алгоритм работы с такими кодами резко упрощается.

Формат 8-разрядного обратного знакового двоичного кода приведен на рисунке 5. На рисунке приведено шесть различных чисел, записанных в этом коде. Слева указаны десятичные числа, соответствующие двоичному представлению обратного кода.

Рисунок 5. Целые обратные двоичные коды

Обратите внимание, что знак числа при инвертировании получается автоматически. При инвертировании положительного числа, в котором знак '+' обозначен логическим нулем, мы получаем логическую единицу, то есть знак '-'.

В этом коде, точно также как и в прямом знаковом двоичном коде можно записывать как 8-разрядные, так и шестнадцати или 32-разрядные двоичные числа. При этом потребуется использовать несколько ячеек оперативного запоминающего устройства, как это иллюстрировалось на рисунке 6 для прямого беззнакового двоичного кода.

Тем не менее, при работе с обратными кодами требуется специальный алгоритм распознавания знака, вычисления абсолютного значения числа, восстановления знака результата числа. Кроме того, в прямом и обратном коде числа для запоминания числа 0 используется два кода тогда, как известно, что число 0 положительное и отрицательным не может быть никогда.

Дополнительные двоичные коды. От перечисленных недостатков свободны дополнительные коды. Эти коды позволяют непосредственно суммировать положительные и отрицательные

числа, не анализируя знаковый разряд и при этом получать правильный результат. Все это становится возможным благодаря тому, что дополнительные числа являются естественным кольцом чисел, а не искусственным образованием как прямые и обратные коды. Кроме того немаловажным является то, что вычислять дополнение в двоичном коде чрезвычайно легко. Для этого достаточно к обратному коду добавить 1. То есть для получения отрицательного числа в дополнительном коде необходимо к модуль числа проинвертировать и прибавить 1.

Рисунок 6. Дополнительные двоичные коды

Диапазон чисел, которые можно записать таким кодом: -128 .. +127. Для шестнадцатиразрядного кода этот диапазон будет: -32768 .. +32767. В восьмиразрядном процессоре для хранения такого числа используется две ячейки памяти, расположенные в соседних адресах.

В памяти объемом n бит может храниться 2^n различных значений. Например, величины типа **Integer** лежат в диапазоне от –32768 (-2^{15}) до 32767 ($2^{15}-1$) и для их хранения отводится 2 байта (16 бит); типа Long — в диапазоне от -2^{31} до $2^{31}-1$ и размещаются в 4 байтах (32 бита).

Кодирование символов.

Набор символов персональных ЭВМ, совместимых с IBM PC, чаще всего является расширением кода ASCII. В этом случае для кодирования символов достаточно одного байта, что позволяет представить 256 символов (с десятичными кодами от 0 до 255). В настоящее время также используются и двухбайтовые

представления символов.

Представление целых чисел в дополнительном коде.

Дополнительный код положительного числа совпадает с его прямым кодом. Прямой код целого числа может быть получен следующим образом: число переводится в двоичную систему счисления, а затем его двоичную запись слева дополняют таким количеством незначащих нулей, сколько требует тип данных, к которому принадлежит число. Например, если число $28_{(10)} = 11100_{(2)}$ объявлено величиной типа **Integer** (шестнадцатибитовое со знаком), то его прямым кодом будет 0000000000011100, а если величиной типа **Long** (тридцатидвухбитовое со знаком), то его прямой код будет 00000000000000000000000000011100. Для более компактной записи чаще используют шестнадцатеричное представление кода. Полученные коды можно переписать соответственно как $001C_{(16)}$ и $0000001C_{(16)}$.

Дополнительный код целого отрицательного числа может быть получен следующим образом:

1) записать прямой код модуля числа;

2) инвертировать его (заменить единицы нулями, нули — единицами);

3) прибавить к инверсному коду единицу.

Например, запишем дополнительный код числа (–28), интерпретируя его как величину типа Long (тридцатидвухбитовое со знаком):

1) прямой код числа 28 есть 00000000000000000000000000011100;

2) инверсный код — 11111111111111111111111111100011;

3) дополнительный код — 11111111111111111111111111100100 или $FFFFFFE4_{(16)}$.

При получении числа по его дополнительному коду, прежде всего, необходимо определить его знак. Если число окажется положительным, то просто перевести его код в десятичную систему счисления. В случае отрицательного числа необходимо выполнить следующие действия:

1) вычесть из кода числа 1;

2) инвертировать код;

3) перевести в десятичную систему счисления. Полученное число записать со знаком минус.

Примеры. Запишем числа, соответствующие дополнитель-

ным кодам.

Дано число 0000000000011111. Поскольку в старшем разряде записан нуль, то результат будет положительным. Это код числа 31.

Дано число 1111111110000001. Здесь записан код отрицательного числа. Вычитаем из кода числа 1:

$1111111110000001_{(2)} - 1_{(2)} = 1111111110000000_{(2)}$.

Инвертируем код: 0000000001111111. Далее переводим инвертированный код в десятичную систему счисления $1111111_{(2)} = 127_{(10)}$. Ответ: –127.

В обратных и дополнительных кодах наблюдается интересный эффект, который называется эффект распространения знака. Он заключается в том, что при преобразовании однобайтного числа в двухбайтное достаточно всем битам старшего байта присвоить значение знакового бита (старшего бита) младшего байта. То есть для хранения знака числа можно использовать сколько угодно старших бит. При этом значение кода совершенно не изменяется.

Использование для представления знака числа двух бит предоставляет возможность контролировать переполнения при выполнении арифметических операций. Рассмотрим несколько примеров. В них число кодируется 8 битами и добавляется дополнительный девятый бит переноса, который должен совпадать с битом знака (восьмым битом).

1) Просуммируем числа 12 и 5

$$\begin{array}{r} 000001100 \\ +\ 000000101 \\ \hline 000010001 \end{array} \Leftrightarrow \begin{array}{r} 12 \\ +\ 5 \\ \hline 17 \end{array}$$

Перенос Знак

В этом примере видно, что в результате суммирования получается правильный результат. Это можно проконтролировать по флагу переноса C, который совпадает со знаком результата (действует эффект распространения знака).

2) Просуммируем два отрицательных числа -12 и -5

$$\begin{array}{r} 111110100 \\ +\ 111111011 \\ \hline 111101111 \end{array} \Leftrightarrow \begin{array}{r} -12 \\ +\ -5 \\ \hline -17 \end{array}$$

В этом примере флаг переноса C тоже совпадает со знаком результата, то есть переполнения не произошло и в этом случае

3) Просуммируем положительное и отрицательное число -12 и +5

$$\begin{array}{r} 11111010\,0 \\ +\;00000010\,1 \\ \hline 11111100\,1 \end{array} \Leftrightarrow \begin{array}{r} -12 \\ +\;5 \\ \hline -7 \end{array}$$

В этом примере при суммировании положительного и отрицательного числа автоматически получается правильный знак результата. В данном случае знак результата отрицательный. Флаг переноса совпадает со знаком результата, поэтому переполнения не было (мы можем убедиться в этом непосредственными вычислениями на бумаге или на калькуляторе).

4) Просуммируем положительное и отрицательное число +12 и -5

$$\begin{array}{r} 00000110\,0 \\ +\;11111101\,1 \\ \hline 00000011\,1 \end{array} \Leftrightarrow \begin{array}{r} +12 \\ -\;5 \\ \hline +7 \end{array}$$

В данном примере знак результата положительный. Флаг переноса совпадает со знаком результата, поэтому переполнения не было и в этом случае.

5) Просуммируем числа 100 и 31

$$\begin{array}{r} 00110010\,0 \\ +\;00001111\,1 \\ \hline 01000001\,1 \end{array} \Leftrightarrow \begin{array}{r} +100 \\ +\;31 \\ \hline +131\text{!!} \end{array}$$

В этом примере видно, что в результате суммирования произошло переполнение восьмибитовой переменной, т.к. в результате операции над положительными числами получился отрицательный результат. Однако если рассмотреть флаг переноса, то он не совпадает со знаком результата. Эта ситуации является признаком переполнения результата и легко обнаруживается при помощи операции «исключающее ИЛИ» над старшим битом результата и флагом переноса. Большинство процессоров осуществляют эту операцию аппаратно и помещают результат во флаг переполнения.

6) Просуммируем числа 100 и 31

$$\begin{array}{r} 11001110\,0 \\ +\;11110000\,1 \\ \hline 10111110\,1 \end{array} \Leftrightarrow \begin{array}{r} -100 \\ -\;31 \\ \hline -131\text{!!} \end{array}$$

В этом примере в результате операции над отрицательными числами при суммировании произошло переполнение

восьмибитовой переменной, т.к. получился положительный результат. И в этом случае, если рассмотреть флаг переноса, то он не совпадает со знаком результата. Отличие от предыдущего случая только в комбинации этих бит. В примере 5 говорят о переполнении результата (комбинация 01), а в примере 6 об антипереполнении результата (комбинация 10).

Умножение двоичных чисел.

Умножить числа 1101_2 и 101_2

Множимое	1101 X	13 X
Множитель	101	5
Частичное произведение	1101	65_{10}
Частичное произведение	0000	
Частичное произведение	1101	
Конечное произведение	1000001_2	

Кодирование вещественных чисел.

Несколько иной способ применяется для представления в памяти персонального компьютера действительных чисел. Рассмотрим представление величин с плавающей точкой.

Любое действительное число можно записать в стандартном виде $M \cdot 10^p$, где $1 \le M < 10$, p — целое. Например, $130900000 = 1{,}309 \cdot 10^8$. Поскольку каждая позиция десятичного числа отличается от соседней на степень числа 10, умножение на 10 эквивалентно сдвигу десятичной запятой на одну позицию вправо. Аналогично деление на 10 сдвигает десятичную запятую на позицию влево. Поэтому приведенный выше пример можно продолжить: $130900000 = 1{,}309 \cdot 10^8 = 0{,}1309 \cdot 10^9 = 13{,}09 \cdot 10^7$. Десятичная запятая «плавает» в числе и больше не помечает абсолютное место между целой и дробной частями.

В приведенной выше записи M называют мантиссой числа, а p – его порядком. Для того чтобы сохранить максимальную точность, вычислительные машины почти всегда хранят мантиссу в нормализованном виде. Мантисса в таком представлении должна удовлетворять условию: $0{,}1s < m < s$. Иначе говоря, мантисса меньше 1 и первая значащая цифра — не ноль (s — основание

системы счисления). При таком представлении запятая будет расположена в мантиссе перед первой значащей цифрой, что при фиксированном количестве разрядов, отведенных под мантиссу, обеспечивает запись максимального количества значащих цифр числа, то есть максимальную точность представления числа в компьютере.

Примеры нормализованного представления чисел:

$$879{,}45 = 0{,}87945 \cdot 10^{5},\ -0{,}0024 = 0{,}24 \cdot 10^{-2}.$$

Способ хранения мантиссы с плавающей точкой подразумевает, что двоичная запятая находится на фиксированном месте. Фактически подразумевается, что двоичная запятая следует после первой двоичной цифры, т.е. нормализация мантиссы делает единичным первый бит, помещая тем самым значение между единицей и двойкой. Место, отводимое для числа с плавающей точкой, делится на два поля. Одно поле содержит знак и значение мантиссы, а другое содержит знак и значение порядка.

Существует стандарт IEEE 754 для представления чисел с одинарной точностью (float) и с двойной точностью (double). Для записи числа в формате с плавающей запятой одинарной точности требуется тридцатидвухбитовое слово. Для записи чисел с двойной точностью требуется шестидесятичетырёхбитовое слово. Чаще всего числа хранятся в нескольких соседних ячейках памяти процессора. Форматы числа в формате с плавающей запятой одинарной точности и числа в формате с плавающей запятой удвоенной точности приведены на рисунке 7.

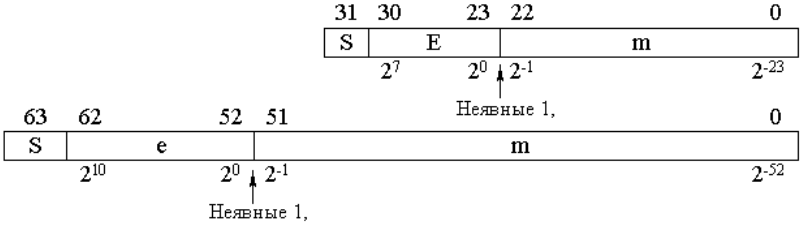


Рисунок 7. Форматы числа в формате с плавающей запятой

На рисунке буквой S обозначен знак числа, 0 – это положительное число, 1 – отрицательное число. ***E*** обозначает смещённый порядок числа. Смещение требуется, чтобы не вводить в число еще один знак. Смещённый порядок всегда положительное число. Для одинарной точности для порядка выделено восемь бит. Для смещённого порядка двойной точности отводится 11 бит. Для одинарной точности смещение принято 127, а для двойной точности —

1023. В десятичной мантиссе после запятой могут присутствовать цифры 1-9, а в двоичной — только 1. Поэтому для хранения единицы после двоичной запятой не выделяется отдельный бит в числе с плавающей запятой. Единица подразумевается, как и двоичная запятая. Кроме того, в формате чисел с плавающей запятой принято, что мантисса всегда больше 1. То есть диапазон значений мантиссы лежит в диапазоне от 1 до 2.

Покажем преобразование действительного числа для представления его в памяти ЭВМ на примере величины типа ***Double***.

Как видно из табл. 1.2, величина типа ***Double*** занимает в памяти 8 байт. На рис. 8 показано, как здесь представлены поля мантиссы и порядка (нумерация битов осуществляется справа налево).

Знак числа	Смещенный порядок					Мантисса			
63	62	61	60	...	52	51	50	...	0

Рис. 8. Представление поля мантиссы и порядка величины типа ***Double***

Можно заметить, что старший бит, отведенный под мантиссу, имеет номер 51, т.е. мантисса занимает младшие 52 бита. Черта указывает здесь на положение двоичной запятой. Перед запятой должен стоять бит целой части мантиссы, но поскольку она всегда равна 1, здесь данный бит не требуется и соответствующий разряд отсутствует в памяти (но он подразумевается). Значение порядка хранится здесь не в дополнительном коде. Для упрощения вычислений и сравнения действительных чисел значение порядка в ЭВМ хранится в виде ***смещенного числа***, т.е. к настоящему значению порядка перед записью его в память прибавляется смещение. Смещение выбирается так, чтобы минимальному значению порядка соответствовал нуль. Например, для типа ***Double*** порядок занимает 11 бит и имеет диапазон от 2^{-1023} до 2^{1023}, поэтому смещение равно $1023_{(10)} = 1111111111_{(2)}$.

Наконец, бит с номером 63 указывает на знак числа.

Таким образом, для получения представления действительного числа в памяти ЭВМ необходимо выполнить следующие действия:

1) перевести модуль данного числа в двоичную систему счисления;

2) нормализовать двоичное число, т.е. записать в виде

$M \cdot 2^p$, где M — мантисса (ее целая часть равна $1_{(2)}$) и p — порядок, записанный в десятичной системе счисления;

3) прибавить к порядку смещение и перевести смещенный порядок в двоичную систему счисления;

4) учитывая знак заданного числа (0 — положительное; 1 — отрицательное), выписать его представление в памяти ЭВМ.

Пример. Запишем код числа –302,25. Двоичная запись модуля этого числа имеет вид 100101110,01. Имеем $100101110{,}01 = 1{,}0010111001 \cdot 2^8$. Получаем смещенный порядок 8+1023=1031. Далее имеем $1031_{(10)} = 10000000111_{(2)}$. Окончательно код числа – 302,25 представим в табл. 1.

Таблица 1. Кодирование числа –302,25

Код	1	10000000111	0010111001	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000	0000000000
Разряд	63	62 ... 52	51 50 ... 0									...

Рассмотрим несколько примеров перевода вещественных чисел из двоичного формата в десятичный.

1) Определить число с плавающей запятой, лежащее в четырёх соседних байтах: 11000001 01001000 00000000 00000000

- Знаковый бит, равный 1 показывает, что число отрицательное.

- Экспонента 10000010 в десятичном виде соответствует числу 130. Вычтя число 127 из 130, получим число 3.

- Мантисса равна: 1,100 1000 0000 0000 0000 0000_2

То есть число равно: = $1{,}1001*2^3{}_2 = 1100{,}1_2 = 12{,}5_{10}$

2) Определить число с плавающей запятой, лежащее в четырёх соседних байтах: 11000011 00110100 00000000 00000000

- Знаковый бит, равный 1 показывает, что число отрицательное.

- Экспонента 10000110 в десятичном виде соответствует числу 134. Вычтя число 127 из 134, получим число 7.

- Мантисса равна: 1,011 0100 0000 0000 0000 0000_2

То есть число равно: = $1{,}0110100*2^7{}_2 = 180_{10}$.

Для того чтобы записать ноль, достаточно записать в смещенный порядок число 00000000_2. Значение мантиссы при этом не имеет значения. Число, в котором все байты равны 0, тоже попадает в этот диапазон значений.

Бесконечность соответствует смещенному порядку 11111111_2 и мантиссе, равной 1,0. При этом существует минус бесконечность и плюс бесконечность (переполнение и антипереполнение), которые часто отображаются на экран монитора как **+INF** и **-INF**. Все остальные комбинации битов (в том числе и все единицы) воспринимаются как не числа и отображаются на экран: **NaN**.

Порядок выполнения работы

1. Осуществить преобразование чисел различных типов в памяти компьютера.

Представить числа в памяти компьютера, соответственно в формате int (16 бит), int (16 бит), double.

Варианты:

1) $96, -200, 34{,}122 \cdot 10^4$
2) $212, -100, 24{,}235 \cdot 10^{-5}$
3) $202, -77, 4{,}165 \cdot 10^{-15}$
4) $35, -42, 1{,}342 \cdot 10^2$
5) $3421, -342, 10{,}135 \cdot 10^4$
6) $121, -122, 30{,}23 \cdot 10^6$
7) $1233, -127, 1{,}67 \cdot 10^{-3}$
8) $765, -9342, 100{,}788 \cdot 10^{14}$
9) $-1321, 742, 8{,}9 \cdot 10^2$
10) $21, -1342, -2{,}435 \cdot 10^4$
11) $131, -4111, -0{,}165 \cdot 10^2$
12) $12342, -88, -16{,}99 \cdot 10^2$
13) $500, -1942, 33{,}543 \cdot 10^{-3}$
14) $-12, 1232, 138{,}555 \cdot 10^{-4}$

2. Сложить первое число со вторым в соответствии с правилами выполнения машинных арифметических операций. Для определения переполнения использовать бит переноса.

3. Умножить два однобайтовых двоичных числа по формуле:

(Последняя цифра номера зачетки+3) * (Предпоследняя цифра номера зачетки+2).

Контрольные вопросы

1. Назовите виды двоичных кодов.
2. В каком диапазоне могут храниться беззнаковые числа в двухбайтовом машинном слове?
3. В каком диапазоне могут храниться знаковые числа в дополнительном двоичном коде в двухбайтовом машинном слове?
4. В каком диапазоне могут храниться знаковые числа в прямом двоичном коде в двухбайтовом машинном слове?
5. Как получить обратный двоичный код числа?
6. Как получить дополнительный двоичный код числа?
7. Как определить возникновение переполнения при сложении чисел?
8. Каков принцип кодирования вещественных чисел?
9. Каким образом осуществляется кодирование вещественных чисел двойной точности?

ЛАБОРАТОРНАЯ РАБОТА 2
ОЗНАКОМЛЕНИЕ С ПАКЕТОМ МОДЕЛИРОВАНИЯ ЭЛЕКТРОННЫХ СХЕМ.

Цель работы: получить основные навыки работы в пакете MultiSim.

Краткие теоретические сведения

1. Описание интерфейса пакета MULTISIM

Одним из наиболее популярных на сегодняшний день средств моделирования и анализа электронных устройств и схем является программный пакет NI MULTISIM. Особенностью пакета является содержание блока логического моделирования цифровых устройств, наличие широкого спектра контрольно-измерительных приборов, максимально приближенных к их промышленным аналогам.

Общий вид интерфейса программы MULTISIM (версия 10.1) представлен на рис. 1. Интерфейс программы можно изменять, с помощью настроек, доступных во *View* командной строки меню. Ниже рассмотрим основные блоки интерфейса, необходимые для достижения цели лабораторной работы[1].

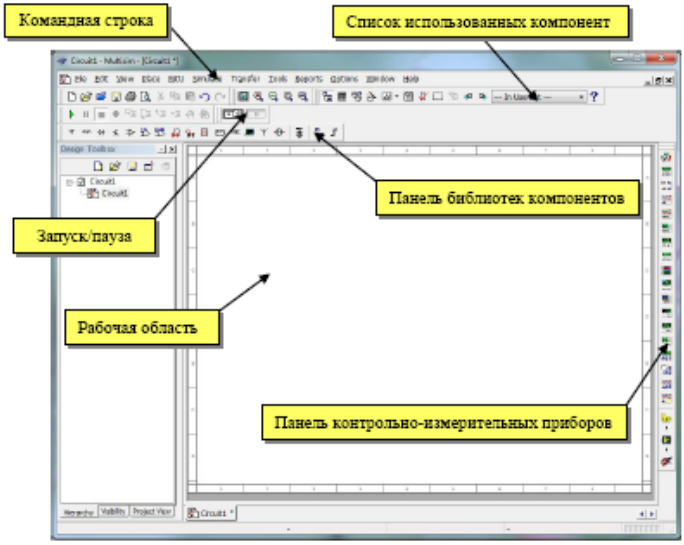


Рис. 1 – Окно схемного редактора программы MULTISIM.

Панель библиотек компонентов (*Components*) содержит:

- источники питания (*Sources*);
- пассивные компоненты и коммутационные устройства (*Basic*);
- диоды (*Diodes*);
- транзисторы (*Transistors*);
- аналоговые микросхемы (*Analog*);
- цифровые микросхемы TTL серии (*TTL*);
- цифровые микросхемы КМОП серии (*CMOS*);
- одиночные цифровые схемы, АЛУ, регистры, счетчики, мультиплексоры, дешифраторы, ОЗУ и т.п. (*Misc Digital*);
- микросхемы смешанного типа (*Mixed*);
- индикаторные устройства (*Indicators*);
- аналоговые вычислительные устройства (*Controls*);
- радиочастотные компоненты (*RF*);
- электромеханические элементы (*Electromechanical*).

Панель контрольно-измерительных приборов (*Instruments*) содержит:

- цифровой мультиметр (*Multimeter*);
- функциональный генератор (*Function Generator*);
- измеритель активной мощности и коэффициента мощности (*Wattmeter*);
- осциллограф (*Oscilloscope*);
- измеритель АЧХ и ФЧХ (*Bode Plotter*);
- генератор слова (*Word Generator*);
- логический анализатор (*Logic Analyzer*);
- логический преобразователь (*Logic Converter*);
- измеритель нелинейных искажений в диапазоне частот от 20 до 200000 Гц (*Distortion Analyzer*);
- спектральный анализатор (*Sperctrum Analyzer*);
- прибор для анализа электрических цепей в обобщенном виде – в виде четырехполюсников, имеющих два входа и два выхода (четыре полюса) (*Network Analyzer*).

При работе с электрическими схемами в пакете MULTISIM предусмотрена возможность применения двух распространенных стандартов изображения компонентов – *ANCI* (США) и

DIN (Европа, Россия). Изменить стандарт доступно в *Options* → *Global Preferences* во вкладке *Parts* (см. рис. 2).

Перед началом работы требуется заранее изменить стандарт на DIN (по умолчанию установлен ANCI). Ниже будут представлены изображения одних и тех же логических элементов в обоих стандартах, однако работать рекомендуется в европейском стандарте, в силу его наиболее широкой применимости.

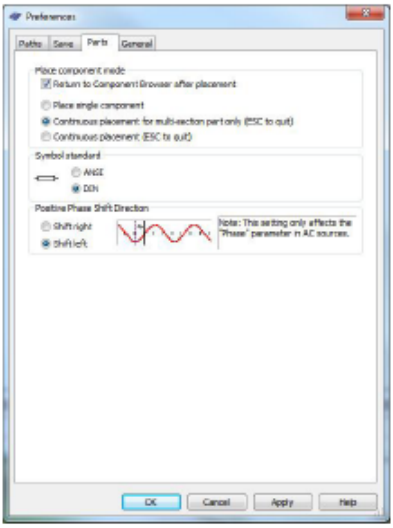


Рис. 2 – Изменение стандарта изображения компонент.

Рабочая область предназначена для построения схем. **Моделирование работы** спроектированной схемы производится путем запуска режима моделирования (*Simulate* → *Run*, или щелчком «тумблера» на верхней панели).

2. Создание новой схемы

Создание новой схемы рассмотрим на примере построения схемы, исследующей логический элемент «И» (рис. 3).

Особенностью данной схемы является её универсальность для последующего исследования других логических элементов. Важно помнить, что цифровые сигналы – это сигналы, имеющие два стабильных уровня – уровень логического нуля и уровень логической единицы. У микросхем, выполненных по различным технологиям, логические уровни могут отличаться друг от друга. В настоящее время наиболее широко распространены две технологии:

ТТЛ (транзисторно-транзисторная логика) и КМОП (комплиментарный металл-оксид-полупроводник). У ТТЛ уровень нуля равен 0,4 В, уровень единицы – 2,4 В. У логики КМОП, уровень нуля очень близок к нулю вольт, уровень единицы – примерно равен напряжению питания.

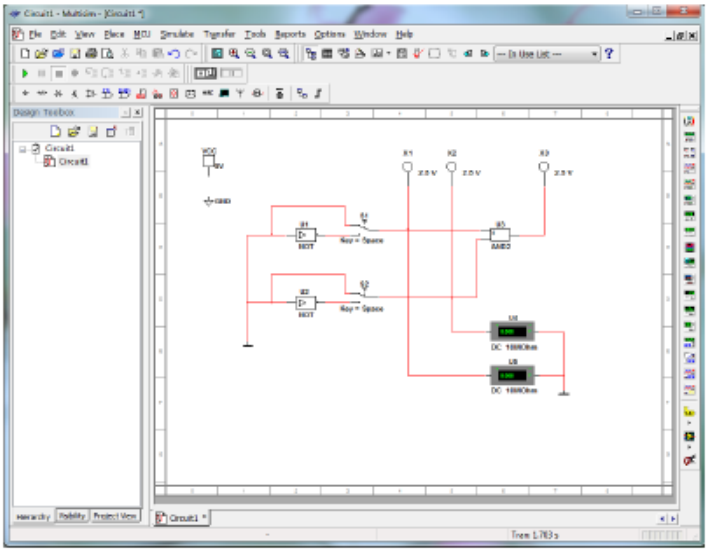


Рис. 3 – Схема исследования логического элемента «И».

Для построения схемы разместим в рабочей области все необходимые компоненты. Для этого воспользуется рассмотренной ранее *Панелью библиотек компонентов* и *Панелью контрольно-измерительных приборов*.

Размещение источников питания

Источник питания подразумевает *источник постоянного напряжения* +5 В (VCC) и *«цифровую землю»* (DGND), которые используют для подачи питания на цифровые компоненты.

1. Нажать на кнопку *Sources* на *Панели библиотек компонентов*. Откроется панель, содержащая источники напряжения или тока. Выбрать источник постоянного напряжения ***VCC***. Курсор примет форму выбранного компонента. Поместить выбранный компонент на схему и щелкнуть левой клавишей мыши по точке схемы, в которой должен быть расположен компонент, изображение компонента появится на схеме.

2. Нажать на кнопку *Sources* на *Панели библиотек компонентов*. Поместить на схему символ «цифровая земля», выбрав DGND.

В пакете MULTISIM помимо заземления для цифровых схем () имеется также имеется обычное *заземление* (), используемое в любых процессах моделирования, за исключением моделирования цифровых устройств в реальном режиме. Обычное заземление может применяться при цифровом моделировании в реальном режиме – с помощью него в рассматриваемом примере можно моделировать источник сигнала логического нуля.

3. Нажать на кнопку *Sources* на *Панели библиотек компонентов*. Поместить на схему символ «заземление», выбрав GROUND.

Размещение переключателей

Для возможности переключения уровня сигнала моделируемой исследовательской установки требуется разместить на схеме два переключателя на два положения.

1. Нажать кнопку *Electromechanical*. Откроется панель с соответствующими компонентами. Выбрать кнопку *SUPPLEMENTARY_CONTACTS*. В окне просмотра выбрать компонент *SPDT_SB* и поместить два образца этого компонента на схему.

Вращать добавленные компоненты возможно кликнув по ним правой кнопкой мыши и выбрать требуемое вращение, либо отображение элемента зеркально по горизонтали (Flip Horizontal) или по вертикали (Flip Vertical).

2. Отобразить добавленные переключатели по горизонтали.

Размещение индикаторов

Индикаторы позволяют отображать наличие или отсутствие сигнала (напряжения) в цепи, а также его электрический уровень. Для первого случая добавим на схему щуп, для второго – вольтметры.

1. Нажать кнопку *Place Indicator*. Откроется панель инструментов индикаторов. Выбрать в семействе PROBE необходимый щуп (probe), представляющий собой светодиод, и поместить три его образца на схему.

2. Нажать кнопку *Place Misc Digital*. В списке компонентов выбрать компонент *2И* (*AND2*), где число указывает необходимое количество входов, и поместить его на схему.

В результате размещения всех необходимых для дальнейшей работы элементов страница схемы будет выглядеть как показано на рис. 5.

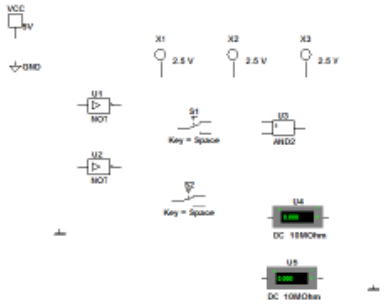


Рис. 5 – Страница схемы с размещенными на ней компонентами.

Для удобства последующего размещения элементов на схеме в дальнейшем можно также воспользоваться **списком использованных компонент** – *In Use List*, куда заносятся применяемые пользователем элементы схемы.

Соединение компонентов проводниками

После того как компоненты размещены на схеме, их необходимо соединить проводниками. Все компоненты имеют выводы, которые используются для этих целей. Для соединения компонент достаточно щелкнуть по выводу требуемого компонента (при этом изображение курсора измениться на крест, показывая, что программа находится в режиме разводки) и провести выводу компонента, к которому необходимо присоединить проводник.

При этом по пути следования проводника можно изменять его направление фиксируя положение проводника в требуемых точках рабочей области.

Для соединения проводников друг с другом нужно добавить точку соединения (*junction*) для чего необходимо выбрать в строке меню команду *Place* → *Junction* и разместить точку в нужное место соединения.

Чтобы удалить проводник, соединяющий два вывода, нужно щелкнуть по проводнику правой клавишей мыши и из всплывающего меню выбрать команду *Delete* или нажать клавишу *Delete*. По завершении соединения проводников должна получиться схема, подобная рис. 3.

Моделирование работы схемы

Процесс моделирования можно запустить или приостановить с помощью команд *Simulation* → *Run* и *Simulation* → *Pause* соответственно. Процесс моделирования можно запускать и останавливать с помощью переключателя-«тумблера».

Порядок выполнения работы.

1. Запустить программу MultiSim.
2. Построить схему, соответствующую формуле
R = X И (Y ИЛИ-НЕ Z) ИЛИ (НЕ X),
и заполнить для нее таблицу истинности вида

X	Y	Z	R
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

3. Построить схему, соответствующую формуле (приведена запись в виде функции fИЛИ-НЕ в связи с тем, что она имеет три аргумента)
R = X И (НЕ(Y) И-НЕ fИЛИ-НЕ (X, Y, НЕ(Z))),
и заполнить для нее таблицу истинности вида

X	Y	Z	R
0	0	0	
0	0	1	
0	1	0	

0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

4. Построить схему для реализации контроля информации с помощью кода Хемминга. Информационная часть содержит 4 бита.

Код Хемминга – это блочный код, позволяющий исправлять одиночные и фиксировать двойные ошибки, разработанный Ричардом Хеммингом в сороковых годах прошлого столетия.

Поскольку информационная часть содержит 4 бита, то количество проверочных бит равно 3 и они будут иметь позиции с порядковыми номерами равными степени двойки: $2^0, 2^1, 2^2$ = 1, 2, 4.

Размещение информационных (И) и проверочных (П) бит в результирующей последовательности будет следующим:

$П_1$	$П_2$	$И_3$	$П_4$	$И_5$	$И_6$	$И_7$
		1		0	0	1

Проверочные биты необходимо определить по следующему правилу:

$$П_1 = И_3 \oplus И_5 \oplus И_7 = 1 \oplus 0 \oplus 1 = 0$$

$$П_2 = И_3 \oplus И_6 \oplus И_7 = 1 \oplus 0 \oplus 1 = 0$$

$$П_4 = И_5 \oplus И_6 \oplus И_7 = 0 \oplus 0 \oplus 1 = 1$$

Таким образом, закодированная последовательность будет иметь следующий вид:

$П_1$	$П_2$	$И_3$	$П_4$	$И_5$	$И_6$	$И_7$
0	0	1	1	0	0	1

После передачи этой последовательности по линии связи на принимающей стороне может быть осуществлена проверка правильности полученной информации и в случае наличия ошибки только в одном бите ее можно исправить.

Для этого необходимо сложить биты

$$E_1 = П_1 \oplus И_3 \oplus И_5 \oplus И_7$$

$$E_2 = П_2 \oplus И_3 \oplus И_6 \oplus И_7 \quad (1)$$

$$E_3 = П_4 \oplus И_5 \oplus И_6 \oplus И_7$$

и число

$$E_1 \cdot 2^0 + E_2 \cdot 2^1 + E_3 \cdot 2^2 \quad (2)$$

равно номеру бита, в котором имеется ошибка.

Например, вместо

$П_1$	$П_2$	$И_3$	$П_4$	$И_5$	$И_6$	$И_7$
0	0	1	1	0	0	1

на приемной стороне получена последовательность

$П_1$	$П_2$	$И_3$	$П_4$	$И_5$	$И_6$	$И_7$
0	0	1	1	0	1	1

В результате вычисления по формуле (1) будет получено

$$E_1 = П_1 \oplus И_3 \oplus И_5 \oplus И_7 = 0 \oplus 1 \oplus 0 \oplus 1 = 0$$

$$E_2 = П_2 \oplus И_3 \oplus И_6 \oplus И_7 = 0 \oplus 1 \oplus 1 \oplus 1 = 1$$

$$E_3 = П_4 \oplus И_5 \oplus И_6 \oplus И_7 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$$

что соответствует ошибке в 6 разряде: $E_1 \cdot 2^0 + E_2 \cdot 2^1 + E_3 \cdot 2^2$ = 6. Таким образом, для исправления этой ошибки необходимо инвертировать бит 6.

Схема имеет вид, представленный на рис. 6.

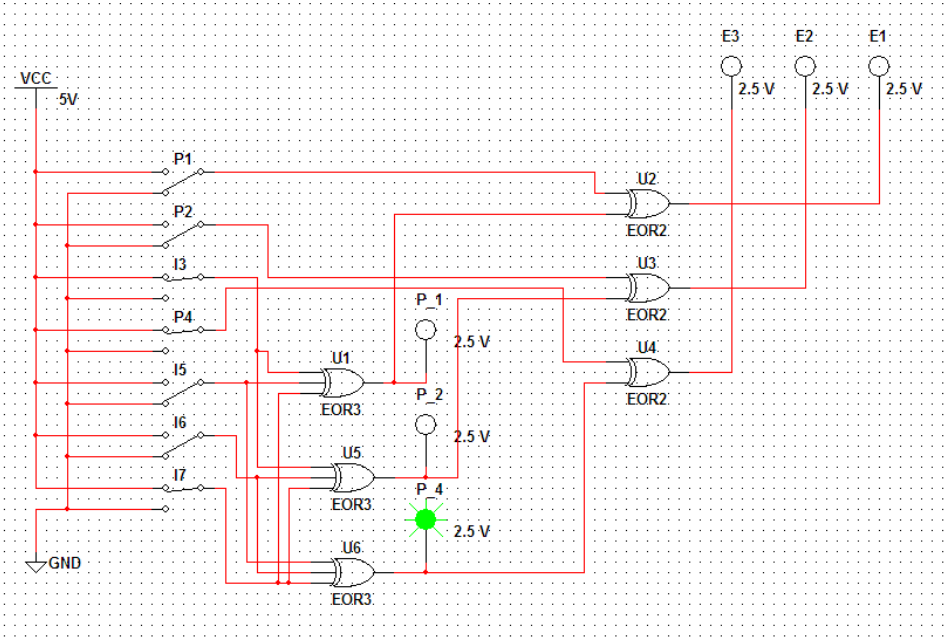


Рис. 6 .

Для ее получения использовались элементы:

– Sources\POWER_SOURCES\VCC (1 шт.),

– Sources\ POWER_SOURCES\DGND (1 шт.),

– Electro_Mechanical\SUPPLEMENARY_CONTACTS\SPDT_SB (7 шт., с применением модификации Flip Horizontal),

– Misc Digital\TIL\EOR3 (Исключающее ИЛИ с тремя входами – 3 шт.),

– Misc Digital\ TIL\EOR2 (Исключающее ИЛИ с двумя входами – 3 шт.),

– Indicators\Probe\PROBE_DIG_GREEN (6 шт.).

На рис. 6 переключатели позволяют установить требуемые значения информационных и проверочных битов, индикаторы P_1, P_2, P_4 показывают требуемые значения корректирующих кодов, индикаторы E1, E2, E3 показывают номер ошибочного бита.

Заполнить таблицу 2, вычислив проверочные биты с помощью схемы, изображенной на рис. 6.

Таблица 2

П1	П2	И3	П4	И5	И6	И7	Внесенная ошибка (№ бита)	Е3	Е2	Е1	Выявленная ошибка (№ бита)
		1		1	0	0	5				
		1		0	1	0	7				
		0		0	0	1	2				
		0		0	1	1	1				
		1		1	0	1	4				
		1		1	0	1	3				
		0		1	1	1	6				

Для каждой схемы внести ошибку в указанный бит и определить с помощью схемы, изображенной на рис. 6, значения Е3, Е2, Е1 и выявите по ним номер ошибочного бита.

5. Задание выполняется двумя студентами. Первый студент тайно от второго вносит ошибку в один бит с помощью переключателя на схеме, изображенной на рис. 6. Второй студент по значениям Е3, Е2, Е1 должен определить номер ошибочного бита.

Контрольные вопросы

1. Каково назначение пакета MultiSim?
2. Какие группы элементов входят в библиотеку компонентов пакета MultiSim?
3. Каким образом кодируются проверочные биты в коде Хемминга?
4. Опишите функционирование схемы на рис. 6.

ЛАБОРАТОРНАЯ РАБОТА 3
СХЕМЫ С ИСПОЛЬЗОВАНИЕМ ТРИГГЕРОВ

Цель работы. Изучить схемы с использованием триггеров. Научиться анализировать цифровые схемы с триггерами.

Краткие теоретические сведения

Триггеры. Основные положения.

Триггер – это устройство последовательностного типа с двумя устойчивыми состояниями равновесия, предназначенное для записи и хранения информации. Под действием входных сигналов триггер может переключаться из одного устойчивого состояния в другое. При этом напряжение на его выходе скачкообразно изменяется с низкого уровня на высокий или наоборот.

Классификация триггеров может быть произведена по принципам логического функционирования и способу восприятия управляющей информации с информационных входов.

По способу восприятия информации различают

– асинхронные триггеры;

– синхронные триггеры.

В асинхронных триггерах воздействие входных информационных (управляющих) сигналов осуществляется непрерывно во времени, а переключение триггера из одного состояния в другое вызывается изменением этих сигналов. В синхронных триггерах воздействие входных сигналов происходит лишь в определенные отрезки времени синхросигнала (в дальнейшем этот вход обозначается буквой *C*).

По способу синхронизации или по виду активных частей синхросигнала, во время которых происходит воздействие входных информационных сигналов и изменение его состояния, различают:

– триггеры, управляемые (тактируемые) импульсом синхронизации;

– триггеры с динамическим управлением (управляемые положительными или отрицательными фронтами. В этом случае восприятие входных сигналов и переключение в новое состояние происходит во время фронта (среза) синхросигнала.

По виду логического функционирования различают триггеры типов

– *RS,*

– *D,*

– *T,*
– *JK* и др.

Асинхронные триггеры. Все асинхронные триггеры можно отнести к разряду примитивных триггерных устройств, так как они являются прозрачными для входных информационных сигналов в любой момент времени и, следовательно, в максимальной степени подвержены действию помех.

Асинхронный триггер типа *RS*. Схема *RS*-триггера на элементах ИЛИ-НЕ и его функциональное описание приведены на рис. 3. Свое название триггер получил (впрочем, как и все остальные триггеры) в соответствии с обозначением своих входов: *S* (*Set* - установка) - вход установки триггера в единичное состояние *Q*=1, *R* (*Reset* – сброс, гашение) – вход установки в состояние *Q*=0.

RS-триггер (рис. 1) – простейший автомат с памятью, который может находиться в двух состояниях. Триггер имеет два установочных входа: установки *S* (*set* – установка) и сброса *R* (*reset* – сброс), на которые подаются входные сигналы от внешних источников. При подаче на установки активного логического уровня триггер устанавливается в единицу ($Q = 1$, $Q' = 0$, здесь штрих означает инвертирование), при подаче активного уровня на вход сброса триггер устанавливается в ноль ($Q = 0$, $Q' = 1$). Если на оба установочных входа подать пассивный логический уровень, то триггер сохраняет предыдущее состояние выходов: $Q = 1$ или $Q = 0$. Каждое состояние устойчиво и поддерживается за счет действия обратных связей. Подача активного уровня одновременно на оба установочных входа запрещена, так как триггер не может быть установлен в ноль и единицу.

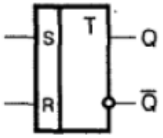


Таблица 1

Входы		Выходы	
R	*S*	*Q*	$\overline{Q}$
0	0	–	
1	0	1	0
0	1	0	1
1	1	нет изменений	

Рис. 1 – RS-триггер.

RS-триггер – основной узел построения последовательных схем. Условия переходов триггеров из одного состояния в другое можно описать табличным, аналитическим или графическим способами. Табличное описание работы RS-триггера на элементах «И-НЕ» представлено в табл. 1.

Синхронные триггеры, тактируемые импульсом. Из синхронных триггеров, тактируемых импульсом, нашли применение два: *RS*- и *D*- триггеры.

Условное графическое обозначение синхронного RS-триггера приведено на рис. 5.

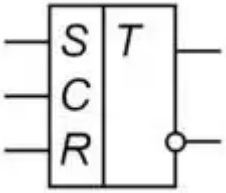


Рис. 5. Условное графическое обозначение синхронного RS-триггера

Синхронный *D*-триггер, тактируемый импульсом.

D-триггер (рис. 2) имеет один информационный вход *D* (*data* – данные) и один счетный вход *C*. Информация с входа *D* записывается в триггер по положительному перепаду импульса на счетном входе и сохраняется до следующего положительного перепада. Кроме счетного *C* и информационного *D* входов, у триггера есть два асинхронных установочных входа *R* и *S*. Установочные входы приоритетные. Активный уровень сигнала на входе *S* устанавливает триггер в состояние единица (*Q*=1), а на входе *R* в состояние ноль (Q=0), независимо от сигналов на остальных входах.

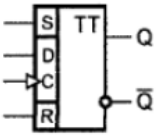


Таблица 2

Входы		Выходы	
D	*C*	*Q*	$\overline{Q}$
1	↑	1	0
0	↑	0	1
*	–	пред. сост.	

Рис. 2 – D-триггер.

Табличное описание работы D-триггера представлено в табл. 2.

Закон функционирования прост:

$Q^{t+1} = D^t$. (1)

Синхронный *JK*- триггер.

JK-триггер (рис. 3) имеет информационные входы J и K, которые по своему воздействию на устройство аналогичны входам S и R синхронного RS-триггера: при $J = 1$ и $K = 0$ триггер по тактовому импульсу C устанавливается в состояние $Q = 1$; при $J = 0$ и $K = 1$ – переключается в состояние $Q = 0$, а при $J = 0$ и $K = 0$ – хранит ранее принятую информацию.

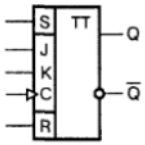


Таблица 3

Входы			Выходы	
J	K	C	Q	$\overline{Q}$
1	1	↓	против. пред.	
1	0	↓	1	0
0	1	↓	0	1
0	0	↓	нет изменений	

Рис. 3 – JK-триггер.

Табличное описание работы JK-триггера представлено в табл. 3.

JK-триггер работает так же как RS-триггер, с одним лишь исключением: при подаче логической единицы на оба входа J и K состояние выхода триггера изменяется на противоположное. Вход J (от англ. Jump — прыжок) аналогичен входу S у RS-триггера. Вход K (от англ. Kill — убить) аналогичен входу R у RS-триггера. При подаче единицы на вход J и нуля на вход K выходное состояние триггера становится равным логической единице. А при подаче единицы на вход K и нуля на вход J выходное состояние триггера становится равным логическому нулю. JK-триггер в отличие от RS-триггера не имеет запрещённых состояний на основных входах, однако это никак не помогает при нарушении правил разработки логических схем.

Порядок выполнения работы

1. Запустить пакет MULTISIM.
2. Провести анализ схемы устранения дребезга контактов.

Анализ предложенных схем, построенных с использованием рассмотренных выше триггеров, позволит на практике применить полученные знания о работе триггеров для понимания работы конкретных схем.

Предварительно рассмотрим ввод в цифровые схемы сигналов (одиночных импульсов произвольной длительности) от механических переключателей (ключи, тумблеры). *Непосредственное ис-*

пользование механического переключателя в качестве источника сигналов для исследования синхронных схем цифровых автоматов использовать нельзя ввиду вибрации механических контактов при переключении.

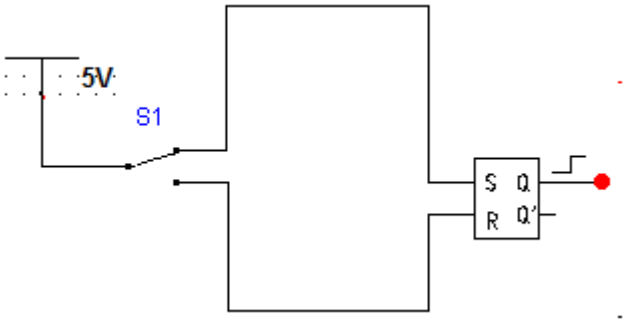


Рис. 4. Схема устранения дребезга контактов

Механическое переключение из-за дребезга контактов приводит к генерации целой последовательности коротких импульсов (с частотой 10...50 Гц), прежде чем установится единичный или нулевой уровень сигнала. Подключение *RS*- триггера так, как это показано на рис. 10, устраняет данную проблему. При нажатии кнопки *S1* схема из положения "0" переходит в положение "1". Во время дребезга контактов, контакт в начальное положение не возвращается, а оказывается в промежуточном положении, тогда на входы поступают "висячие" единицы и триггер находится в состоянии хранения. Дребезг устраняется. Длительность импульса определяется временем нажатия кнопки переключателя *S1*.

В MultiSim схему (рис. 5) можно реализовать на RS-триггере (Misc Digital/TIL/SR_FF). В схеме также используется ключ *S1* SPDT_SB.

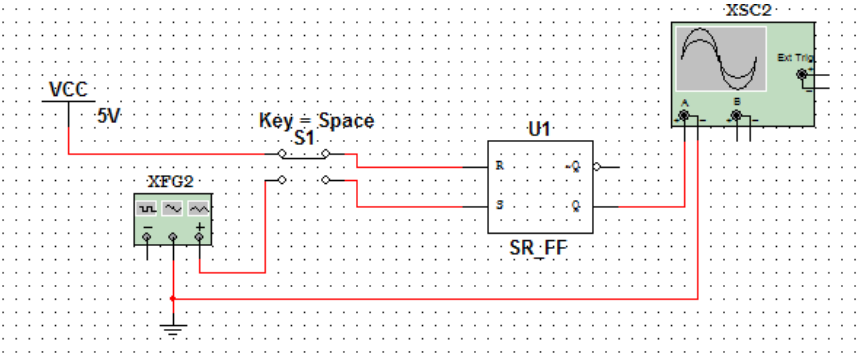


Рис. 5. Модель схемы, устраняющей дребезг контактов.

При отключенном положении ключа (в верхнем положении) на R-вход триггера подается единица, а на S-вход – ноль, что переводит триггер в нулевое состояние. При переключении ключа в положение включено на S-вход триггера подаются импульсы с генератора *XFG1*, имитирующие дребезг контактов, что приводит триггер в единичное состояние. Генератор импульсов имеет свойства, изображенные на рис. 6.

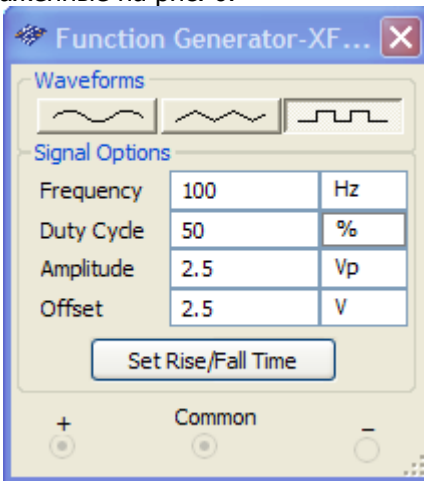


Рис. 6. Свойства генератора импульсов.

3. Исследовать триггерные схемы, построенные на базе D- и JK-триггеров (рис. 7).

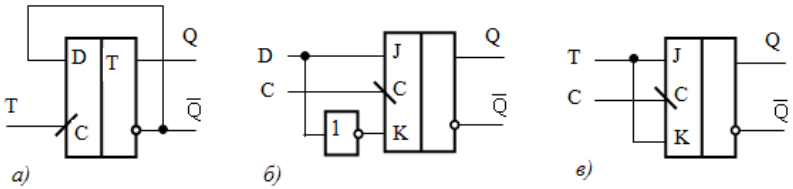


Рис. 7. Схемы асинхронного D-триггера (а) и синхронных JK-триггеров (б, в).

Рассмотрим пример исследования триггерной схемы, изображенной на рис. 7, б.

Для этого разместите на схеме JK-триггер (Misc Digital/TIL/JK_FF), элемент И-НЕ (Misc Digital/TIL/NAND2) для моделирования инвертора, источник питания (Sources/POWER_SOURCES/VCC) и цифровую землю (Sources/POWER_SOURCES/DGND), генератор импульсов (Instruments/Function Generator) и осциллограф (Instruments/Oscilloscope).

На *C*-вход триггера будем подавать импульсы с генератора, настроив его, как показано ранее на рис.6. На информационные входы будем подавать сигналы с ключа (рис. 8) или генератора слов (рис. 9).

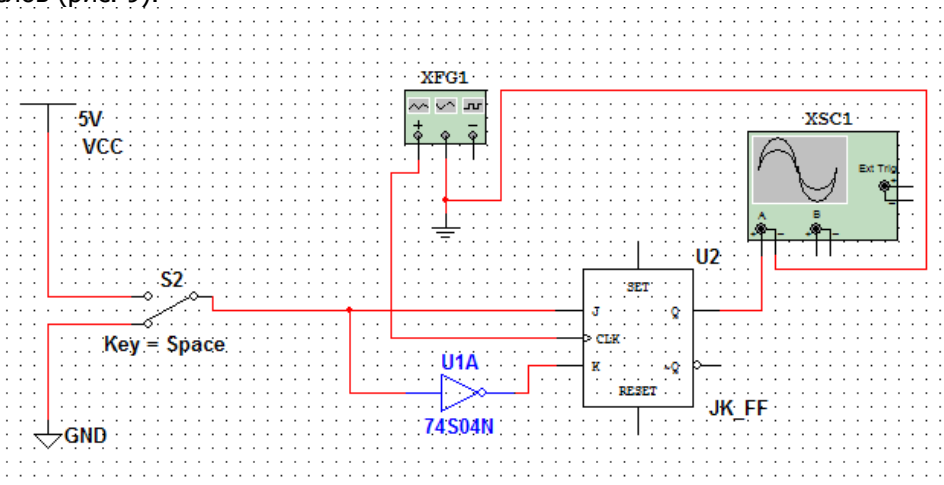


Рис. 8.

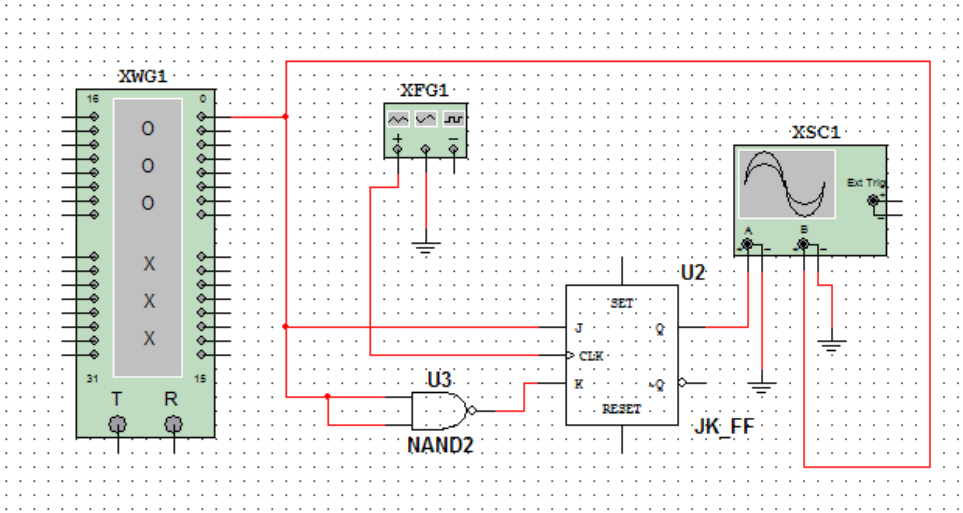


Рис. 9.

Генератор слов настроен на подачу сигналов 00010110010 на 0-выход, как показано на рис. 10.

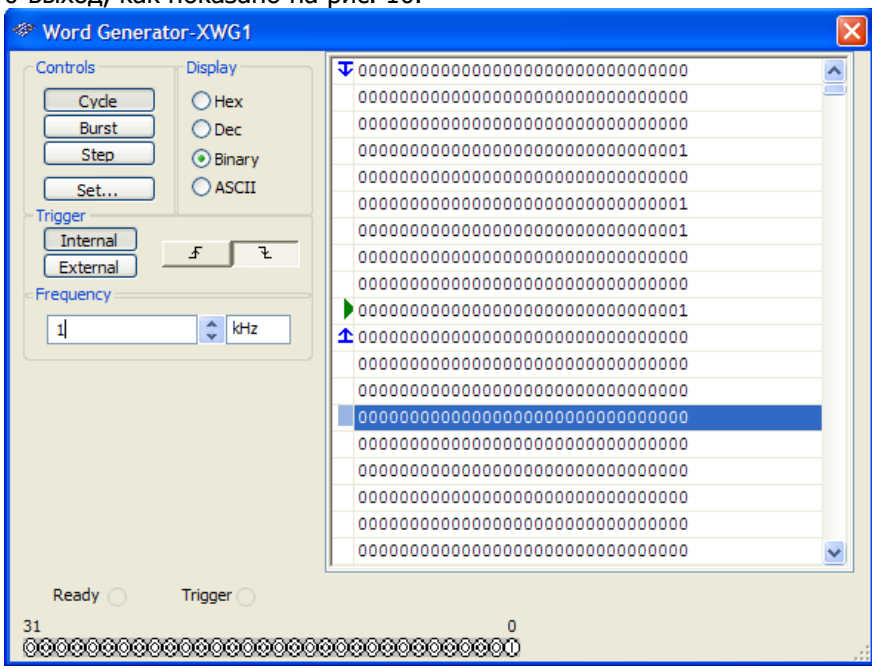


Рис. 10.

Как видно из рис. 11, полученного на осциллографе, выход триггера повторяет сигналы на входе с некоторым отставанием.

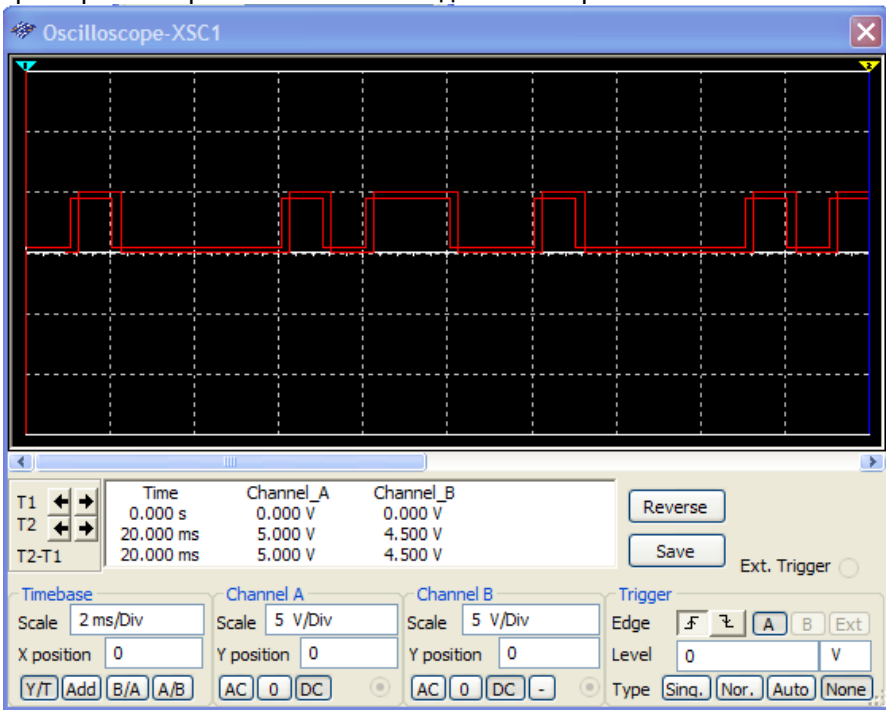
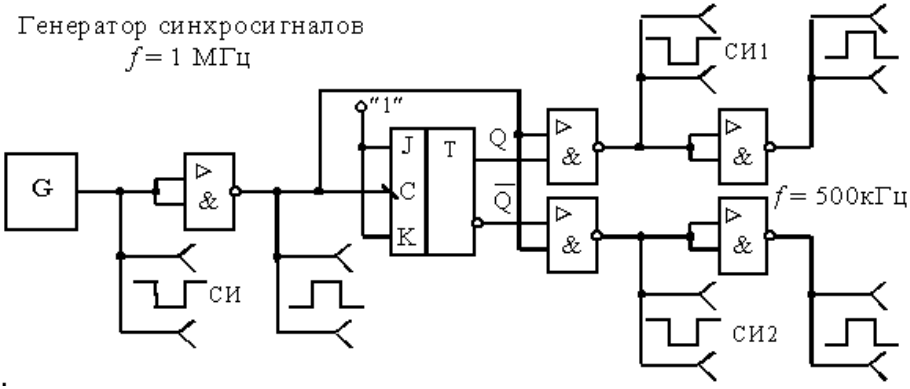


Рис. 11.

Аналогично исследуйте схемы на рис. 7, а и 7, в.

3. Исследовать схему преобразования синхропоследовательности в двухфазную последовательность.

Работа цифровых устройств, содержащих двоичные элементы памяти на триггерах, сопровождается передачей данных по тракту их обработки от предшествующего блока к последующему. Такая передача данных строго регламентируется во времени синхросигналами, разрешающими прием и передачу данных для каждого блока. Простейшая схема двухфазной последовательности синхросигналов изображена на рис. 12. в которой распределение синхросигналов основной последовательности синхроимпульсов (СИ) (f = 1 МГц) осуществляется по двум фазам синхронизации СИ1 и СИ2. Обратите внимание, что *конъюнкция синхросигналов СИ1 и СИ2 в любой момент времени равна "0" — важный принцип двухфазной синхронизации.*



.

Рис. 12. Схема формирования двухфазной синхропоследовательности (*а*) и временные диаграммы ее работы (*б*).

Осуществить моделирование приведенной схемы в MultiSim и убедиться в уменьшении частоты на выходе в два раза и наличии двухфазной синхронизации.

4. Исследовать схему формирования двух последовательностей импульсов со сдвигом на четверть периода относительно друг друга (рис. 13).

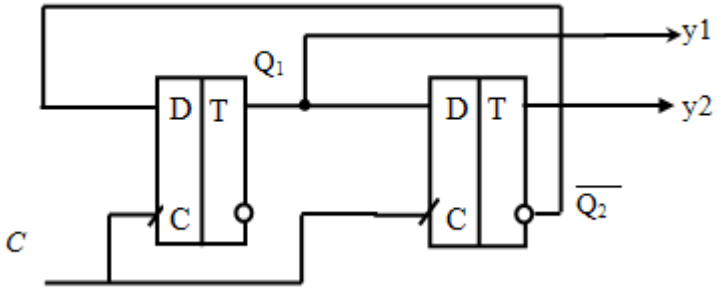


Рис. 13. Схема формирования двух последовательностей импульсов
со сдвигом на четверть периода относительно друг друга

Данная схема описывается следующими формулами в моменты времени $k+1$:

$$C^{k+1} = \overline{C}^{k}$$

При переходе *C* из 0 в 1, поскольку срабатывание триггеров осуществляется по нарастанию синхроимпульса:

$$Q_1^{k+1} = \overline{Q}_2^k$$

$$\overline{Q}_2^{k+1} = \overline{Q}_1^k$$

$$y_1^{k+1} = Q_1^{k+1}$$

$$y_2^{k+1} = Q_2^{k+1}$$

При переходе *C* из 1 в 0 *Q*1 и *Q*2 не меняются.

Таким образом, составим таблицу переключения выходов *Q*1, *Q*2 вручную. Для этого возьмем синхроимпульсы и проанализируем переключение триггеров на каждом такте (таблица 1).

Таблица 1.

Номер такта	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
C	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0		
*y1 = Q*1	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0
$\overline{Q2}$	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
*y2 = Q*2	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1

Наблюдается сдвиг *Q*2 относительно *Q*1 на четверть периода (длительность периода – 8 тактов – выделен подчеркнутым шрифтом в таблице 1, сдвиг – 2 такта).

Осуществить моделирование приведенной схемы в MultiSim и убедиться в наличии сдвига на четверть периода. D-триггер реализовать с помощью элемента Misc Digital/TIL/D_FF.

5. Исследовать схему синхронизатора внешнего одиночного импульса произвольной длительности.

Внешний сигнал (командный сигнал) часто имеет смысл некоторого одиночного события, временное начало и протяжённость которого ничем не регламентируются. Учитывая, что синхронные цифровые устройства правильно воспринимают входные сигналы только в определённые моменты времени, необходимо осуществлять привязку внешних сигналов к синхропоследовательности, действующей в системе. Одна из схем, решающая подобную задачу, приведена на рис. 14.

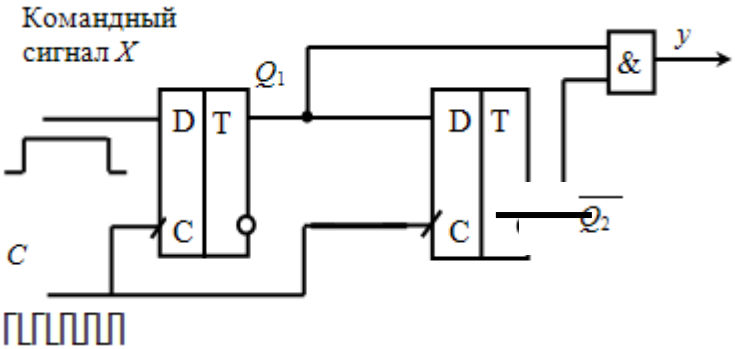


Рис. 14. Схема формирования импульса, равного периоду синхросигнала

Данную схему описывают формулы:

Если *C* переключается из 0 в 1, то

$$\overline{Q}_2^{k+1} = Q_1^k$$

$$Q_1^{k+1} = D_1^k = X^k$$

$$y^{k+1} = Q_1^{k+1} \; И \; \overline{Q}_2^{k+1}$$

Если *C* переключается из 1 в 0, то *Q*1 и *Q*2 не меняются.

Аналогично примеру выше заполните вручную строки 3-6 таблицы 2.

Таблица 2

№ такта	1	2	3	4	5	6	7	8	9	10	11	12	13	14
C	0	1	0	1	0	1	0	1	0	1	0	1	0	1
X	0	0	1	1	1	1	1	1	0	0	0	0	0	0
*Q*1														
*Q*2														
y														

Убедитесь, что синхроимпульсы на выходе схемы появляются, только когда командный сигнал *X* имеет единичное значение.

Промоделируйте полученную схему с использованием пакета MultiSim.

Выясните, как изменится работа схемы, если в цепь синхронизации второго триггера включить инвертор (показан пунктиром на рис. 15).

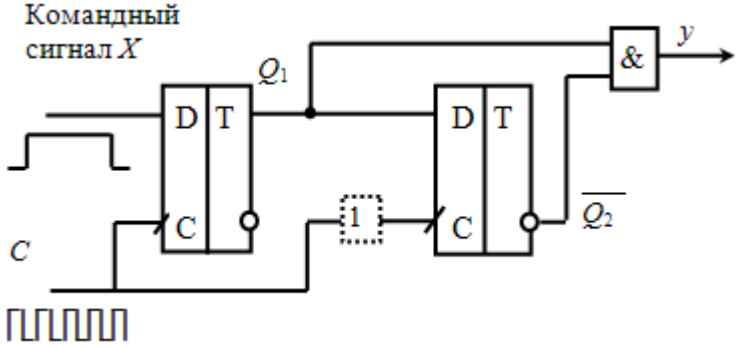


Рис. 15. Исследуемая схема с инвертором

6. Исследовать схему синхронизатора внешнего одиночного импульса с дополнительной функцией генерации пачки импульсов.

Схема на рис. 16 сложнее предыдущей ввиду выполнения дополнительной функции (длительность пачки импульсов равна длительности командного импульса). Расположение и длительность командного сигнала КС относительно синхропоследовательности СИ – произвольное.

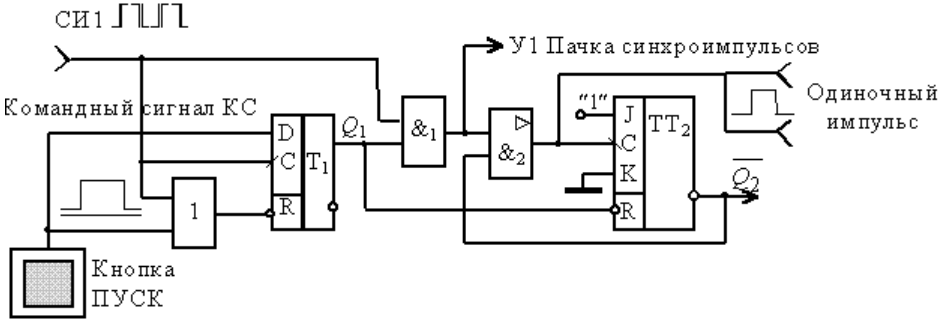


Рис. 16. Формирователь одиночного синхроимпульса одновременно
с пачкой синхросигналов.

Промоделировать в MultiSim схему на рис. 16. Для моделирования T1 использовать элемент Misc Digital/TIL/D_FF, а для TT2 – Misc Digital/TIL/JK_FF_NEGR (рис. 17).

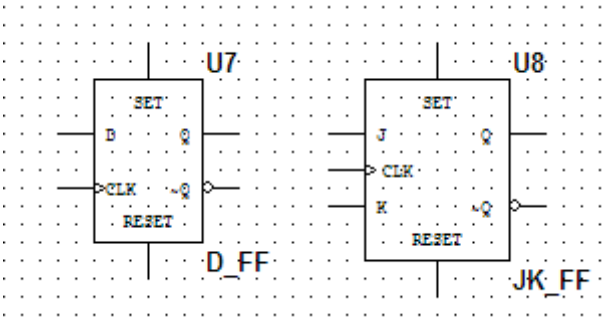


Рис. 17. Внешний вид триггеров в библиотеке MultiSim

7. Реализовать бегущую строку с использованием RS-триггеров (рис. 18) в MultiSim.

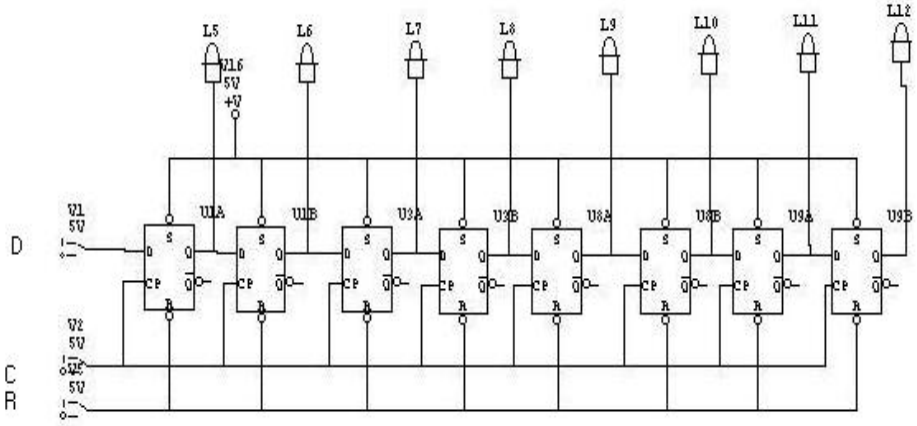


Рис. 18. Схема бегущей строки с использованием триггеров.

На вход необходимо подавать сигналы с ключа или генератора слов и убедиться, что сигналы, подаваемые на вход, последовательно пробегают по светодиодам в виде бегущей строки. На вход D подавать сигналы можно с помощью Генератора слов (Instruments/Word Generator) или переключателя (Electro_Mechanical/SUPPLEMENARY_CONTACTS/SPDT_SB). Для визуальной демонстрации частоту синхроимпульсов необходимо выбрать достаточно низкой 1–5 Гц.

Контрольные вопросы

1. Назовите основные особенности триггеров?
2. Дайте классификацию триггеров.
3. В чем отличие между асинхронными и синхронными триггерами?
4. В чем отличие между RS- и JK-триггерами?
5. Опишите функционирование схемы устранения дребезга контактов на рис. 4.
6. Каким образом функционирует синхронных D-триггер?

ЛАБОРАТОРНАЯ РАБОТА 4
СХЕМЫ С ИСПОЛЬЗОВАНИЕМ СУММАТОРОВ И СЧЕТЧИКОВ

Цель работы: Знакомство с принципом действия сумматоров и счетчиков различных типов, освоение методики синтеза синхронных счетчиков с произвольным модулем счета, ознакомление с особенностями работы типовых счетчиков в интегральном исполнении.

Краткие теоретические сведения

Сумматоры

На рисунке 1 приведена таблица истинности сумматора по модулю 2.

X	Y	Out
0	0	0
0	1	1
1	0	1
1	1	0

Рисунок 1. Таблица истинности сумматора по модулю 2.

В соответствии с принципами построения произвольной таблицы истинности получим схему сумматора по модулю 2. Эта схема приведена на рисунке 2.

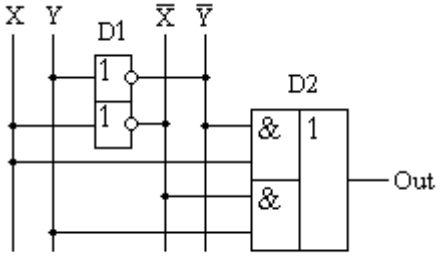


Рисунок 2. Принципиальная схема, реализующая таблицу истинности сумматора по модулю 2.

Сумматор по модулю 2 (схема исключающего "ИЛИ") изображается на схемах как показано на рисунке 3.

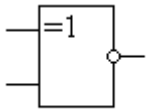


Рисунок 3 Изображение схемы, выполняющей логическую функцию исключающего "ИЛИ".

Сумматор по модулю 2 выполняет суммирование без учета переноса. В обычном двоичном сумматоре требуется учитывать перенос, поэтому требуются схемы, позволяющие формировать перенос в следующий двоичный разряд. Таблица истинности такой схемы, называемой полусумматором, приведена на рисунке 4.

A	B	S	PO
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Рисунок 4. Таблица истинности полусумматора.

В соответствии с принципами построения произвольной таблицы истинности получим схему полусумматора. Эта схема приведена на рисунке 5.

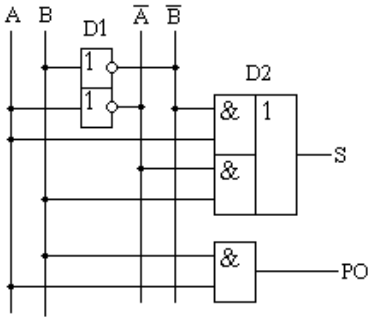


Рисунок 5. Принципиальная схема, реализующая таблицу истинности полусумматора.

Полусумматор изображается на схемах как показано на рисунке 6.

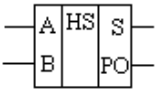


Рисунок 6 Изображение полусумматора на схемах.

Схема полусумматора формирует перенос в следующий разряд, но не может учитывать перенос из предыдущего разряда, поэтому она и называется полусумматором. Таблица истинности полного двоичного одноразрядного сумматора приведена на рисунке 7.

PI	A	B	S	PO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Рисунок 7. Таблица истинности полного двоичного одноразрядного сумматора.

В соответствии с принципами построения схемы по произвольной таблице истинности получим схему полного двоичного одноразрядного сумматора. Эта схема приведена на рисунке 8.

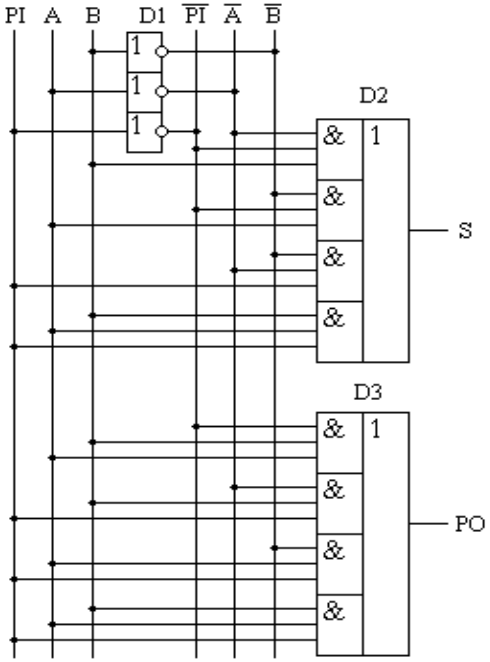


Рисунок 8. Принципиальная схема, реализующая таблицу истинности полного двоичного одноразрядного сумматора.

Полный двоичный одноразрядный сумматор изображается на схемах как показано на рисунке 9.

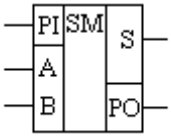


Рисунок 9 Изображение полного двоичного одноразрядного сумматора на схемах.

Для того, чтобы получить многоразрядный сумматор, необходимо соединить входы и выходы переносов соответствующих двоичных разрядов. Схема соединения приведена на рисунке 10.

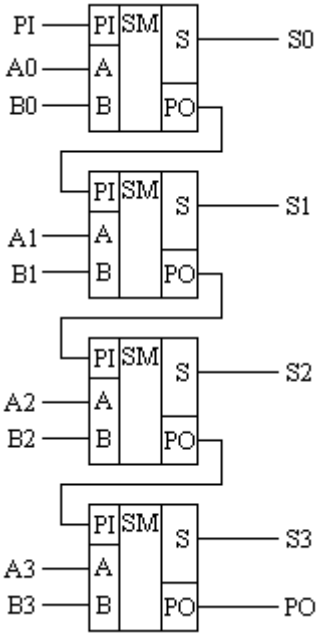


Рисунок 10. Принципиальная схема многоразрядного двоичного сумматора.

Полный двоичный многоразрядный сумматор изображается на схемах как показано на рисунке 11.

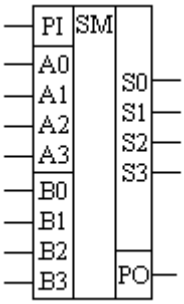


Рисунок 11. Изображение полного двоичного многоразрядного сумматора на схемах.

Порядок выполнения работы

1. Запустить пакет MultiSim

2. Осуществить построение схемы проверки отсутствия ошибок при передаче удвоенного байта данных.

При передаче данных возможно искажение информации. Пусть устройство передает информацию побайтово, причем в четном байте содержится передаваемая информация, а в нечетном — ее копия. Выход схемы должен быть равен 0, если информация получена без ошибок.

Построить схему, определяющую правильность полученной информации и выдающую на выход 0 в случае, если байты равны. Для проверки равенства двух байтов предлагается использовать сумматор, который будет складывать нечетный байт и дополнительный код четного байта. При сложении будет получен байт равный 0, если числа равны и байт не равный 0, если числа не равны. За схемой сложения необходимо добавить схему, которая формирует бит, равный 0, если ошибки нет и 1 в противном случае. Восьмибитный сумматор реализовать с помощью элементов **Misc Digital/TIL/Full Adder**.

3. Построить схему проверки отсутствия ошибок при передаче удвоенной тетрады данных в одном байте.

Пусть устройство передает информацию побайтово, причем в каждом байте содержится только 4 бита полезной информации (0-3), а биты 4-7 являются повтором. Построить схему, определяющую правильность полученной информации путем сложения младшей и дополнительного кода старшей декады декад байта с помощью

сумматора. Выход схемы должен быть содержать бит, равный 1, если ошибки нет.

При сложении будет получено 4 бита = 0000_2, если числа равны и не равные 0000_2, если числа не равны. За схемой сложения необходимо добавить схему, которая формирует бит, равный 1, если ошибки нет и 0 в противном случае. Четырехбитный сумматор реализовать с помощью элементов Misc Digital/TIL/Full Adder.

Контрольные вопросы

1. Какие виды сумматоров знаете? В чем их отличие?
2. Как выглядит таблица истинности полного одноразрядного сумматора?
3. Как построить многоразрядный сумматор на базе одноразрядных полных?

ЛАБОРАТОРНАЯ РАБОТА 5
СХЕМЫ С ИСПОЛЬЗОВАНИЕМ РЕГИСТРОВ

Цель работы: изучение принципа работы и построения схем триггерных регистров.

Краткие теоретические сведения

Наиболее распространенным узлом аппаратных средств вычислительной техники являются регистры. Регистры строятся на базе синхронизированных RS , D и JK-триггеров. По способу приема и выдачи информации регистры делятся на следующие типы:

- с параллельным приемом и выдачей (рис.1, а);
- с последовательным приемом и выдачей (рис.1, б);
- с последовательным приемом и параллельной выдачей (рис.1, в);
- с параллельным или последовательным приемом и последовательной выдачей (рис.1, г);
- комбинированные, с различными способами приема и выдачи (рис.1, д) и реверсивные.

Регистры с параллельным приемом и выдачей информации часто служат для хранения информации и называются регистрами памяти или хранения. Изменение хранящейся информации в регистре памяти (запись новой информации) осуществляется после установки на входах D_0 . . . D_m новой цифровой информации и при поступлении определенного уровня или фронта синхросигнала на “C” входе регистра. Количество разрядов записываемой цифровой информации определяется количеством триггеров, образующих регистр. Этот асинхронный установочный вход называют входом R для “сброса” триггеров регистра в ноль независимо от состояний уровня сигнала на входе C.

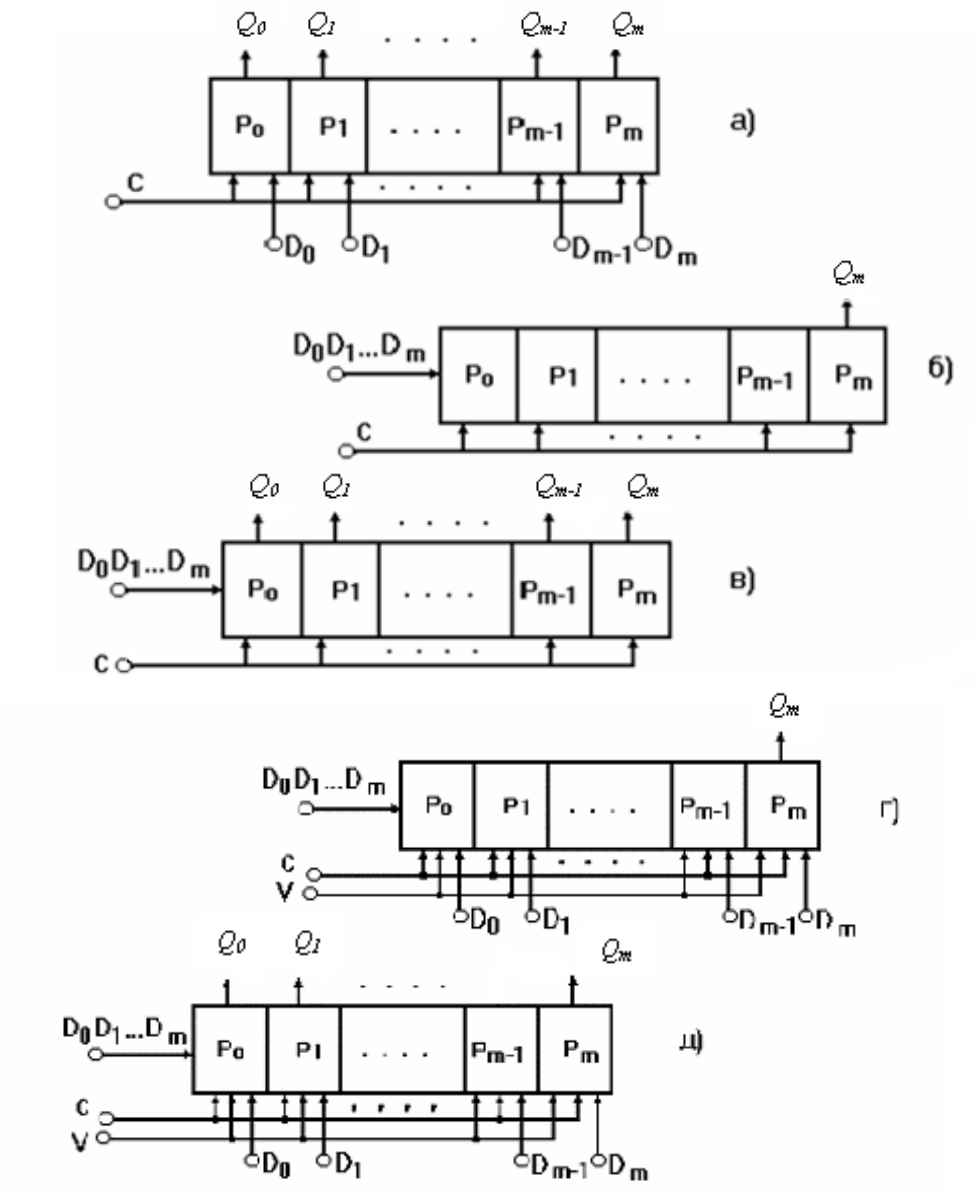


Рис. 1. Функциональные схемы основных типов регистров

На рис. 2 приведены схемы четырехразрядных регистров памяти на D- и RS-триггерах, синхронизируемых уровнем и фронтом синхроимпульсов (обычно четыре триггера объединены в одном корпусе интегральной схемы).

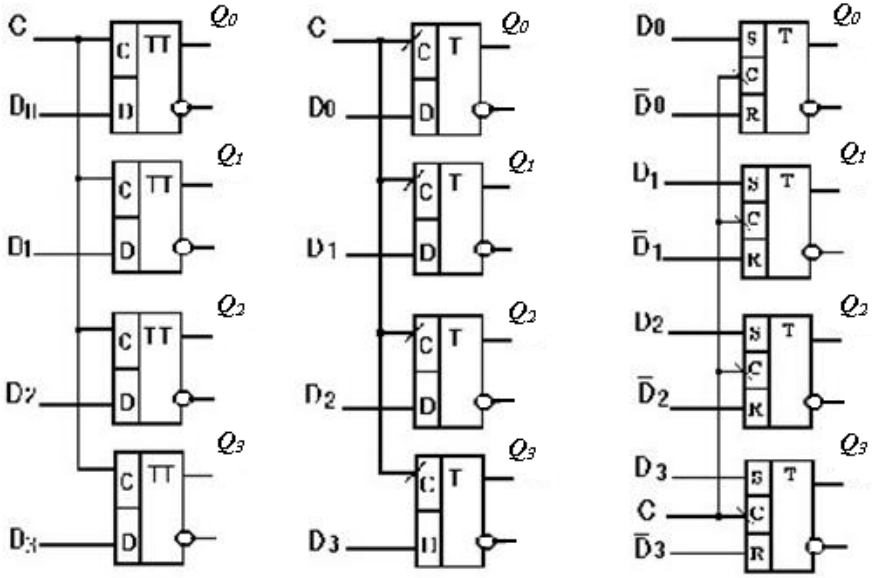


Рис. 2. Регистры хранения, на D – триггерах, синхронизируемых уровнем синроимпульса (а), передним фронтом (б) и на RS – триггерах, синхронизируемых задним фронтом (в)

При отсутствии синхросигнала *C* регистр хранит информацию, при наличии *C* – он воспринимает входные сигналы и устанавливает их значения на выходах при появлении соответствующего фронта. Обычно информация, снимаемая с его выходов, не изменяет предыдущего состояния регистра.

Регистры с последовательным приемом или выдачей информации называются регистрами сдвига. Регистры сдвига могут выполнять функции хранения и преобразования информации. Они

могут быть использованы для построения умножителей и делителей чисел двоичной системы счисления, т.к. сдвиг двоичного числа влево на один разряд соответствует умножению его на два, а сдвиг вправо – делению на два. Регистры сдвига широко используются для выполнения различных временных преобразований цифровой информации: накопление последовательной цифровой информации с последующей одновременной выдачей (преобразование последовательного кода в параллельный код) или одновременный прием информации с последующей последовательной выдачей (преобразование параллельного кода в последовательный). Регистры сдвига могут служить также в качестве элементов задержки данных, представленных в цифровой форме. Так m-разрядные регистры с последовательным приемом и выводом осуществляют задержку передачи информации на m тактов машинного времени.

Регистры сдвига проще реализуются на D-триггерах (рис. 3, а) или на RS-триггерах (рис. 3, б), где для ввода информации в первый разряд используется инвертор (первый разряд преобразуется в D-триггер).

Для избегания неверных переключений разрядов регистра все регистры сдвига строятся на базе двухступенчатых триггеров или триггеров синхронизируемых фронтом синхроимпульса. Параллельный вывод информации из регистра сдвига (см. рис. 3, в) осуществляется съёмом данных с прямых выходов всех триггеров регистра к отдельным выводам (на рис. 3, а и б эти выводы показаны штриховыми линиями). На рис. 3 приведены схемы четырехразрядных регистров сдвига, реализованных на D- и RS-триггерах, а временные диаграммы, поясняющие работу регистра сдвига на D-триггерах (рис. 3, в), приведены на рис. 4, где изменение состояний триггеров происходит по переднему фронту синхросигнала.

Запись новой информации в триггеры регистра происходит в течение очень короткого времени - за время длительности фронта синхроимпульса. Обычно оно равно времени переключения триггера в новое состояние.

Работу регистра сдвига рассмотрим на примере схемы, приведенной на рис. 13, а. Пусть в начале все триггеры регистра находятся в состоянии логического нуля, т.е. $Q_0=0$, $Q_1=0$, $Q_2=0$, $Q_3=0$. Если на входе D-триггера T_1 имеет место логический 0, то поступление синхроимпульсов на входы “C” триггеров не меняет их состояния.

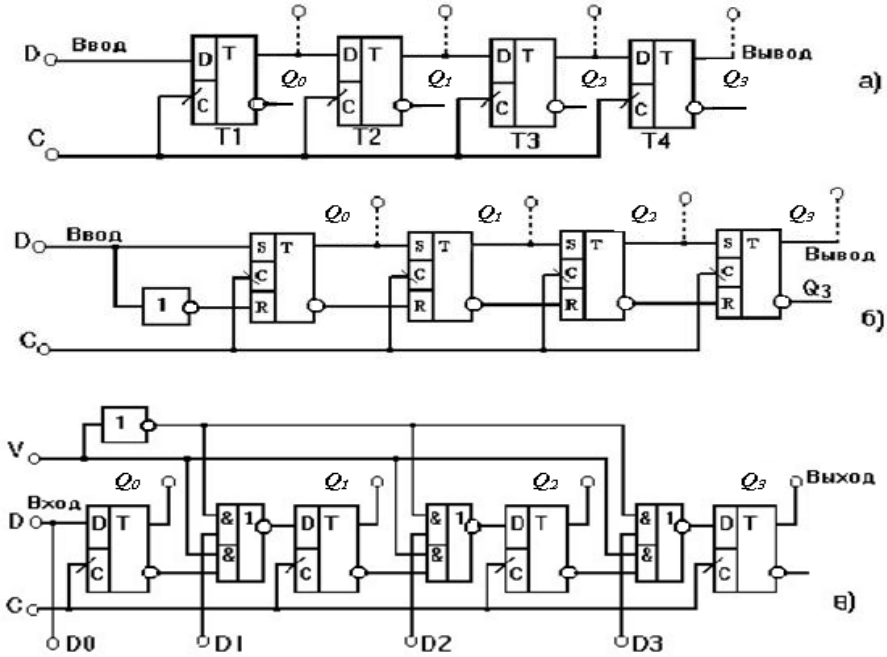


Рис. 3. Последовательные регистры на D – триггерах (а), RS – триггерах (б) и универсальный регистр на D – триггерах с левым сдвигом на один разряд (в)

Как следует из рис. 3, а, синхроимпульс поступает на соответствующие входы всех триггеров регистра одновременно и записывает в них то, что имеет место на их информационных входах D. Так как на информационных входах триггеров Т2, Т3, Т4 - уровни логического “0”, а на вход “D” первого триггера, по условию примера, подается “0” из внешнего источника информации, то состояние регистра не меняется.

При подаче на вход “D” (рис. 3, а) первого триггера “1”, с приходом первого за ним переднего фронта синхроимпульса, в этот триггер запишется “1”, а в остальные триггеры - “0”, т.к. к моменту поступления синхроимпульса на их входах ещё присутствовал логический “0”. Таким образом, в каждый триггер записывается та ин-

формация (тот бит), которая была на его входе "D" в момент поступления переднего фронта синхроимпульса.

При поступлении переднего фронта второго синхроимпульса логическая "1" , с выхода первого триггера, запишется во второй триггер, и в результате происходит сдвиг первоначально записанной "1" с триггера Т1 в триггер Т2, из триггера Т2 в триггер Т3 и т.д. (рис. 4). Таким образом, производится сдвиг поступающей на вход регистра информации в последовательном коде на один разряд влево (L1) в каждом такте синхроимпульса, если T_1 хранит младший разряд числа, а T_4 – старший и выходы Q_0, Q_1, Q_2, Q_3 имеют веса 2^0, 2^1, 2^2, 2^3 при хранения целого числа (при отсутствии импульсов синхронизации). Сдвиг влево (L) – это сдвиг в сторону старших разрядов числа, сдвиг вправо (R) – в сторону младших.

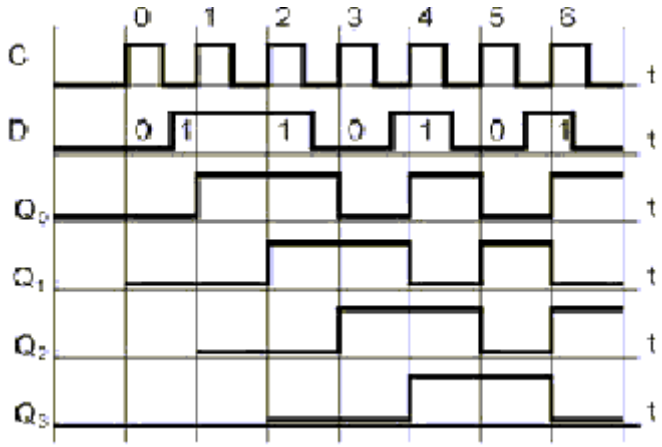


Рис. 4. Операция записи данных в регистр

После поступления m синхроимпульсов (на рис. 4 m=4), регистр оказывается полностью заполненным разрядами числа, вводимого через вход "D". В течение следующих четырех синхроимпульсов производится последовательный поразрядный вывод числа из Q_3 триггера T_4, после чего регистр оказывается полностью очищенным (при условии подачи на его вход уровня "0" в режиме вывода числа), т.е. обнулен. Такой сдвиг называется арифметическим, когда при выводе информация теряется. Если крайний разряд реги-

стра Q_3 соединить со входом первого триггера Т, то сдвиг будет происходить циклически, без потери данных.

На рис. 3, б все триггеры будут также производить сдвиг числа влево на один разряд L1 на задний фронт в момент ухода импульса синхронизации. Особенностью регистра на рис. 3, в является то, что он имеет управляющий вход V. При V=1 он аналогичен регистру на рис. 3, а, а при V=0 он может параллельно принимать четырехразрядное число $D_0D_1D_2D_3$, а затем параллельно его выдавать с выходов Q_0, Q_1, Q_2, Q_3.

На базе регистров сдвига можно построить кольцевые счетчики - счетчики Джонсона. Счетчик Джонсона имеет коэффициент пересчета, вдвое больший числа составляющих его триггеров. В частности, если счетчик состоит из трех триггеров (m=3), то он будет иметь шесть устойчивых состояний. Счетчик Джонсона используется в системах автоматики, например, в качестве распределителей импульсов.

Таблица состояний счетчика Джонсона содержит 2m (m - количество триггеров в регистре) строк и m-столбцов. Для построения кольцевого счетчика достаточно соединить инверсный выход с младшим входом "D" первого триггера, задействовав циклический сдвиг. Схема трехразрядного счетчика Джонсона, реализованного на D-триггерах приведена на рис. 5, а.

а)

N	Q_2	Q_1	Q_0
1	0	0	1
2	0	1	1
3	1	1	1
4	1	1	0
5	1	0	0
6	0	0	0

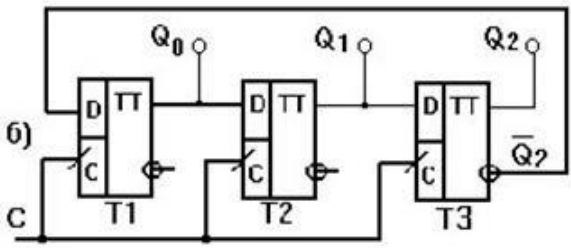


Рис. 5. Таблица состояний а) и схема б) счетчика Джонсона на трехразрядном регистре сдвига

Предположим, что вначале все триггеры (рис. 5, б) находятся в состоянии "0", т.е. $Q_0= Q_1=Q_2=0$. При этом на входе "D" первого триггера присутствует уровень "1", т.к $\overline{Q}_2 = 1$. Первым синхроим-

пульсом в триггер Т1 запишется "1", вторым - единица запишется в первый триггер, из первого - во второй и т.д. до тех пор, пока на всех выходах регистра не будет "1". После заполнения регистра единицами, на инверсном выходе триггера Т3 появится $\overline{Q}_2$ = 0 и четвертым синхроимпульсом в Т1 запишется логический "0".

После поступления последующих трех синхроимпульсов регистр обнуляется и на его вход "D" снова подается уровень "1". Таким образом, цикл повторения состояния кольцевого счетчика состоит из шести тактов синхросигнала. Как видим, при работе в начале от первого триггера до последнего триггера распространяется "волна единиц", а затем "волна нулей".

Порядок выполнения работы

1. Запустить программу MultiSim.

2. Построить четырехразрядный регистр сдвига в соответствии с вариантом:

Вариант 1 - на базе RS-триггера;
Вариант 2 - на базе D-триггера;
Вариант 3 - на базе JK-триггера;

Рассмотрим для примера построение четырехразрядный регистр сдвига.

На панели инструментов «Components» выбираем «Misc Digital»(рис.6).

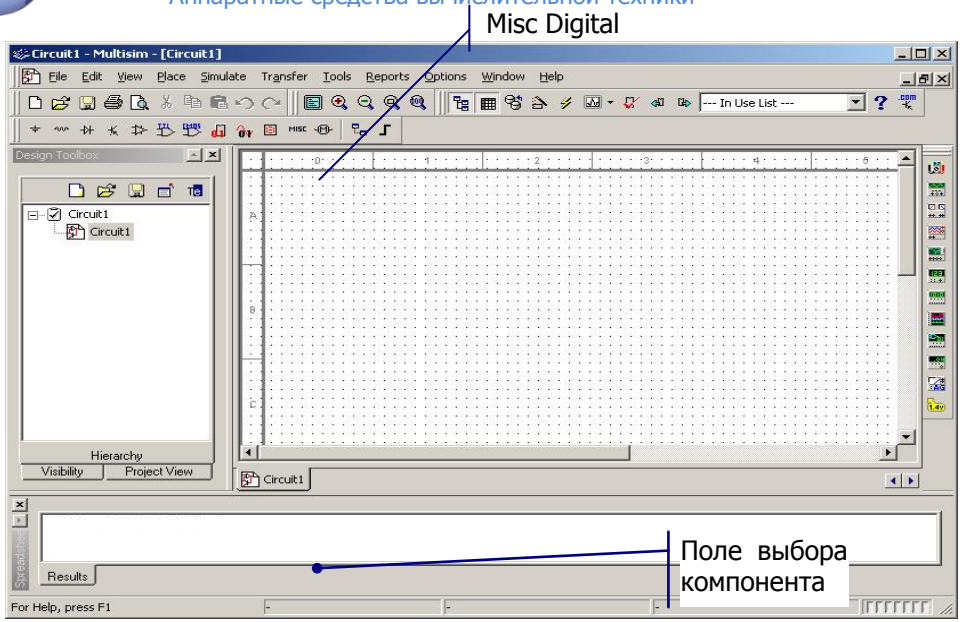


Рис. 6. Окно программы Multisim с новым проектом Circuit1

Для построения схемы требуется элемент Misc Digital/TIL/D_FF. Строим требуемую схему (Рис. 7).

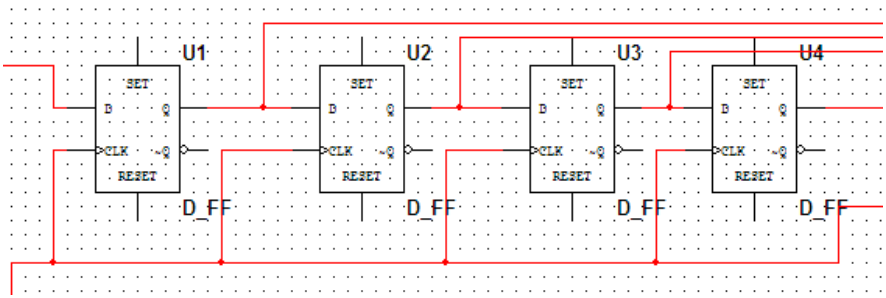


Рис. 6. Схема четырехразрядного регистра сдвига

3. Установить генератор и инструменты.

На правой панели инструментов «Instruments» выбираем Word Generator и Logic Analizer (или в строке меню: Simulate -> instruments-> Word generator (Logic Analizer)). Эти элементы также следуют за курсором мыши до клика на рабочем поле. Размещаем их, и соединяем со схемой (рис. 7).

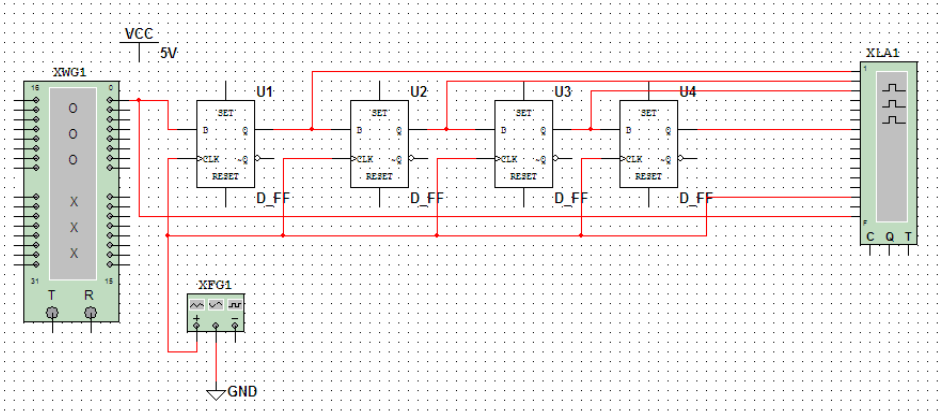


Рис. 7. Схема исследования четырехразрядного регистра сдвига

Делаем двойной щелчок мыши на инструменте «Word Generator» и задаем входные комбинации и частоту (рис. 8).

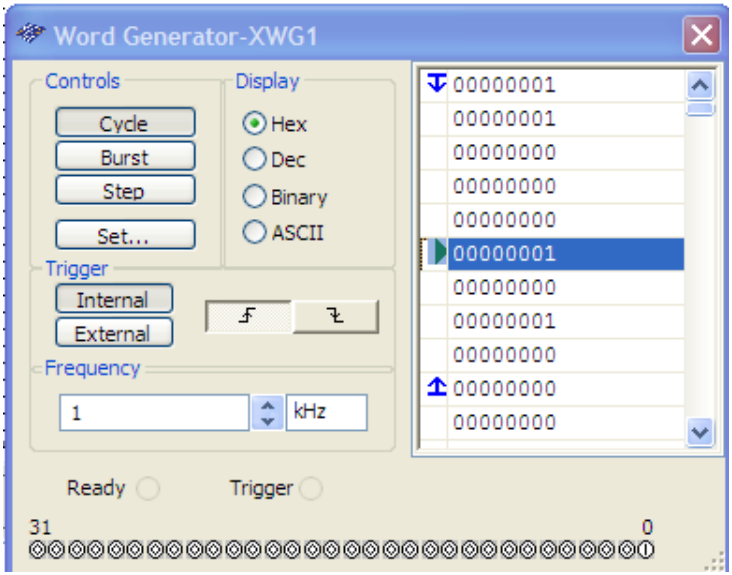


Рис. 8. Настройки инструмента «Word Generator»

Для инструмента «Logic Analyzer» задаем следующие свойства (рис. 9).

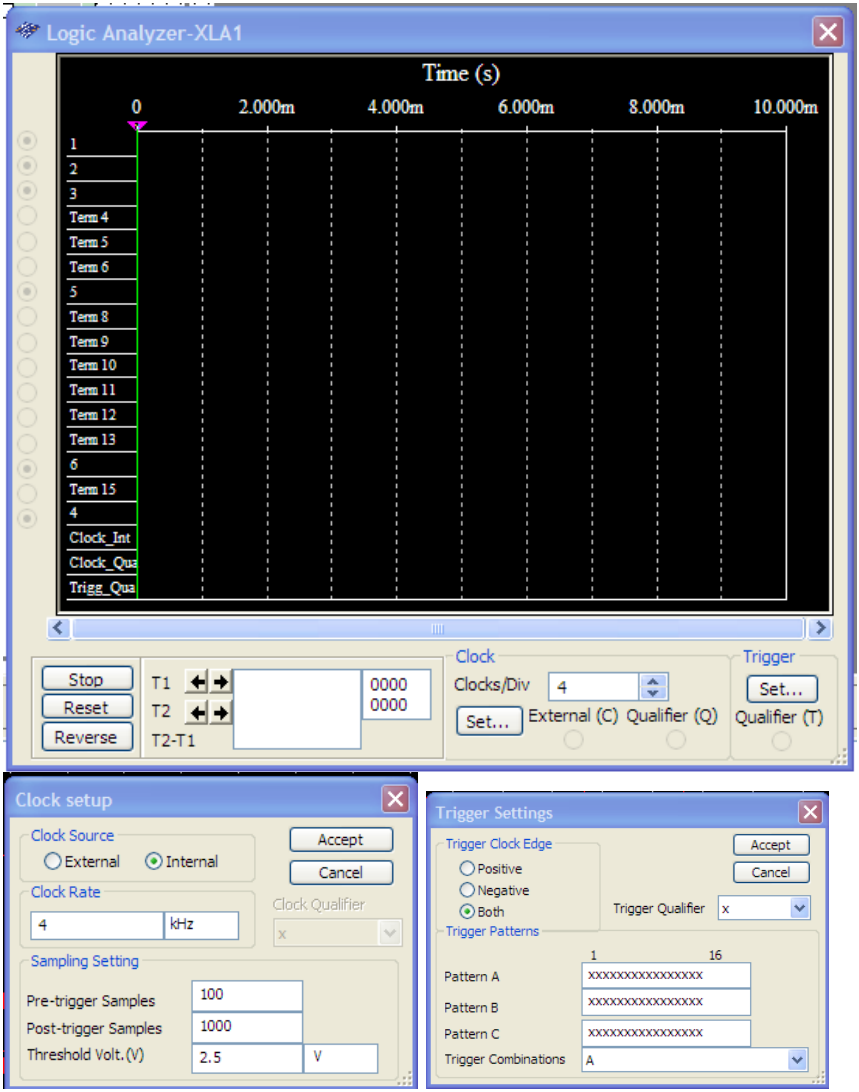


Рис. 9. Настройки инструмента «Logic Analyzer»

4. Промоделировать схему и построить диаграмму работы регистра.

Как видно из графика (рис. 10), выходной сигнал регистра сдвига повторяет входной сигнал с отставанием на 4 такта.

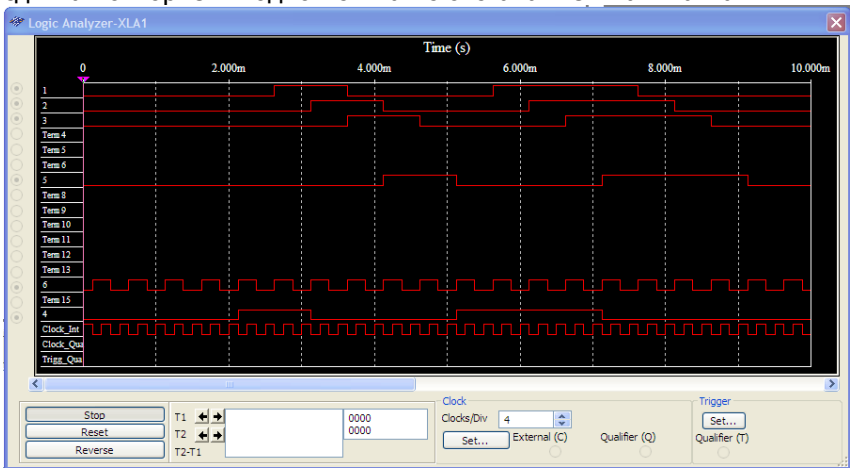


Рис. 10. Результат моделирования

5. Составить отчет.

Отчет по работе должен содержать:

- Описание принципа работы регистра сдвига;
- Функциональную схему регистра на триггерах, согласно варианту;
- Временные диаграммы работы регистра сдвига;
- Выводы

Контрольные вопросы

1. На базе каких элементов можно построить регистр?
2. По каким признакам классифицируются регистры?
3. Чем определяется разрядность регистров?
4. Назовите назначение регистров.
5. Объяснить принцип работы последовательного регистра.
6. Объяснить принцип работы параллельного регистра.
7. Объяснить принцип работы последовательно-параллельного регистра.
8. Объяснить принцип работы параллельно-последовательного регистра.

ЛАБОРАТОРНАЯ РАБОТА 6
ИССЛЕДОВАНИЕ АРИФМЕТИКО-ЛОГИЧЕСКОГО УСТРОЙСТВА

Цель работы. Исследовать особенности и принцип функционирования арифметико-логического устройства.

Краткие теоретические сведения

Арифметико-логическое устройство (АЛУ) – вычислительный блок, который под управлением устройства управления служит для выполнения арифметических и логических преобразований над данными, называемыми *операндами*. Разрядность операндов обычно называют размером машинного слова.

Для изучения принципов действия АЛУ рассмотрим в пакете MULTISIM микросхему SN74181N (аналог отечественной микросхемы **К155ИП3**), представленную на рис. 4. Микросхема 74181 содержит арифметико-логическое устройство (АЛУ), с помощью которого можно выполнить 16 логических и 16 арифметических операций над 4-разрядными операндами (A0 и B0).

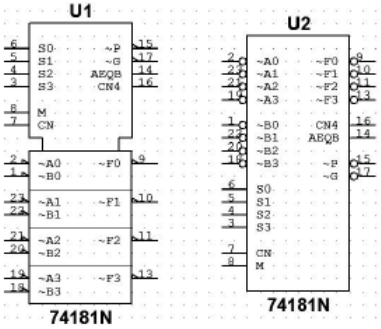


Рис. 4 – Микросхема 74181 в стандарте DIN (*слева*) и ANSI (*справа*).

1,2 – информационные входы младшего разряда операндов B0 и A0, соответственно;

3,4,5,6 – набор входов «выбор функции» (S3,S2,S1,S0);

7 – вход «перенос» (CN);

8 – вход «режим работы» (M);

9 – младший разряд выходной образованной функции (F0);

10,11 – выходы образованной функции (F1,F2);

13 – старший разряд выходной образованной функции (F3);

14 – выход «сравнения» (AEQB);

15 – выход «распространения переноса» (P);

16 – выход «перенос» (CN4);

17 – выход «образование переноса» (G);

18,19 - информационные входы старшего разряда операндов B3 и A3, соответственно;

20,21,22,23 – информационные входы операндов B2, A2, B1, A1, соответственно.

Операнды A и B поступают на соответствующие входы микросхемы 74181 (активный уровень напряжения — низкий). Род работы АЛУ выбирается с помощью входа M (режим работы): логические операции выполняются при подаче на вход M напряжения высокого уровня (1), а арифметические — при подаче напряжения низкого уровня (0). Затем, согласно кодовой таблице (табл. 1), на входах S0-S4 выбирается необходимая для выполнения функция, а результат получают на выходах F0-F3 (активный уровень напряжения — низкий).

Микросхема 74181 используется также в качестве компаратора. Если операнды одинаковы, то на выходе A = B формируется напряжение высокого уровня.

Таблица 1 – Кодовая таблица микросхемы 74181.

Код функции				**Активный уровень низкий**	
S_0	S_1	S_2	S_3	**Арифметические (M = L, C_n = L)**	**Логические (M = H)**
0	0	0	0	A минус 1	$\overline{A}$
1	0	0	0	AB минус 1	$\overline{AB}$
0	1	0	0	$A\overline{B}$ минус 1	$\overline{A}+B$
1	1	0	0	минус 1 (доп. до 2)	лог. 1
0	0	1	0	A плюс $(A+\overline{B})$	$\overline{A+B}$
1	0	1	0	A плюс $(A+\overline{B})$	$\overline{B}$
0	1	1	0	A минус B минус 1	$\overline{A}\oplus\overline{B}$
1	1	1	0	$A+\overline{B}$	$A+\overline{B}$
0	0	0	1	A плюс $(A+B)$	$\overline{AB}$
1	0	0	1	A плюс B	$A\oplus B$
0	1	0	1	$A\overline{B}$ плюс $(A+B)$	B
1	1	0	1	$A+B$	$A+B$
0	0	1	1	A плюс A $(2\times A)$	лог. 0
1	0	1	1	A плюс AB	$A\overline{B}$
0	1	1	1	A плюс $A\overline{B}$	AB
1	1	1	1	A	A

Порядок выполнения работы

Соберите схему для исследования АЛУ (рис. 10). Микросхема 74181 расположена в разделе типовых схем ТТЛ: *Place TTL → 74181N*. Поскольку активным уровнем на выходе микросхемы является низкий уровень напряжения, то для удобства работы следует подключить к каждому выходу логические элементы НЕ и световые индикаторы (*X*1-*X*4, для отображения состояния выходов *F*0-*F*3, соответственно).

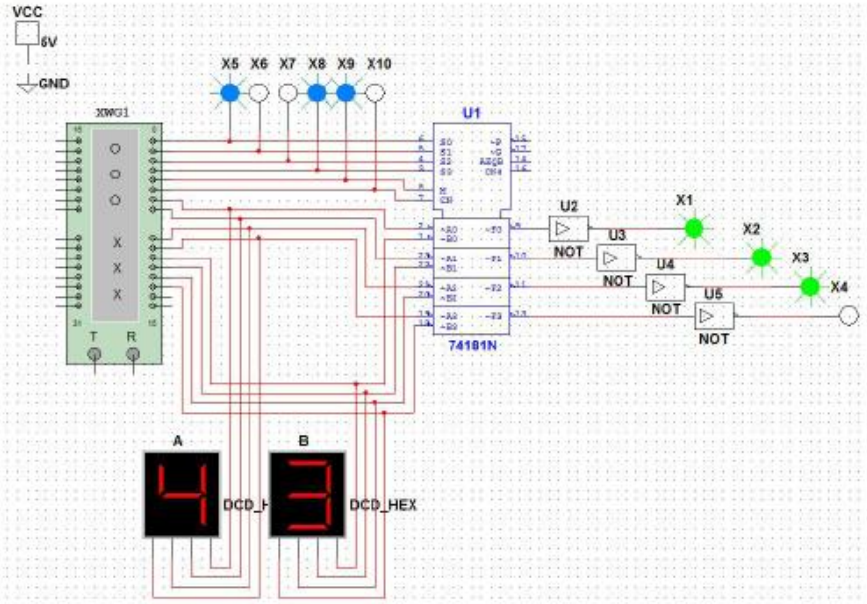


Рис. 10 – Схема исследования микросхемы АЛУ 74181.

Цепочка световых индикаторов *X5-X10* индицирует состояния входов выбора функции S0, S1, S2, S3, M, CN. Ко входам информационных сигналов, определяющих операнды A (A0-A3) и B (B0-B3) подключите семисегментные индикаторы (DCD_HEX).

Настройте генератор слов (рис. 11). Согласно задаваемому на вход коду управления и значения операндов записываются в виде:

$$\underbrace{0011}_{B}\underbrace{0100}_{A}\underbrace{0}_{CM}\underbrace{11001}_{S3\text{-}S0}$$

1 группа 4 бит – четырёхразрядное двоичное слово B (B3,B2,B1,B0);

2 группа 4 бит – четырёхразрядное двоичное слово A (A3,A2,A1,A0);

3 группа 1 бит – одноразрядный сигнал переноса C (CN);

4 группа 1 бит – одноразрядный сигнал управления M (1 – логические операции, 0 – арифметические операции);

5 группа 4 бит – управляющий код (S3,S2,S1,S0).

Изменяя управляющий код (группа 5) и значение сигнала управления (группа 4) можно выбрать любую операцию, которую данное АЛУ может выполнять (табл. 1).

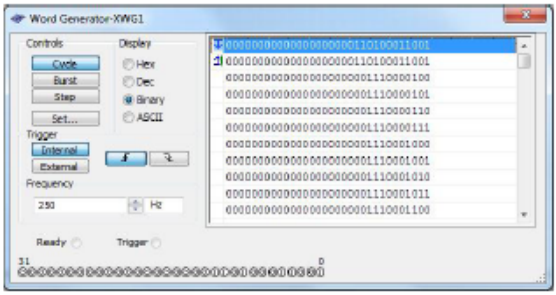


Рис. 11 – Задание кода в генераторе слов.

1. Арифметические операции.

Задайте операнды A = 4 (0100) и B = 3 (0011). Согласно табл. 1, определите для АЛУ код, позволяющий выполнить операции:

– сложения A плюс B;

– A минус B минус 1;

– вывод A;

– A минус 1.

На основе полученных данных заполните таблицу 2.

Табл. 2 – Результаты исследования АЛУ

Операнд B				Операнд A				Перенос	Тип операции	Управляющий код				Состояние выходов				Описание операции
B3	B2	B1	B0	A3	A2	A1	A0	C	M	S3	S2	S1	S0	F3	F2	F1	F0	A+B
																		A–B–1
																		A
																		A–1

*2. **Логические операции.***

Настройте АЛУ в качестве компаратора (сравнивающее устройство). Задайте операнды A = 4 (0100) и B = 3 (0011). Согласно табл. 1, определите для АЛУ код, позволяющий сравнить данные операнды ($A \oplus B$).

Результаты занесите в отчет.

Контрольные вопросы

1. Какие входные сигналы подаются на входы АЛУ?
2. Какие операции может выполнять АЛУ?
3. В каком формате в АЛУ подаются команды и операнды?

ЛАБОРАТОРНАЯ РАБОТА 7
СХЕМЫ С ИСПОЛЬЗОВАНИЕМ МИКРОСХЕМ ПАМЯТИ

Цель работы: изучить способы подключения микросхем памяти в микропроцессорном устройстве с выбором адресного пространства.

Краткие теоретические сведения

Память состоит из ячеек, каждой из которых присваивается свой адрес. Совокупность адресов, которые могут быть сформированы процессором, образует *адресное пространство МПС.* Адреса памяти могут занимать все адресное пространство МПС (АП) или его часть, а сама память независимо от ее технической реализации может быть условно представлена набором регистров (ячеек), число которых M, а разрядность — N (рис. 1).

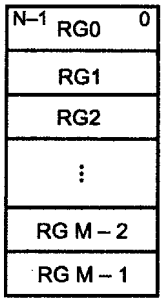


Рис. 1. Условное представление памяти

Свои адреса имеют и внешние устройства (ВУ). Процессор при обмене данными всегда должен выбрать только одну из ячеек памяти или одно ВУ. Такой выбор осуществляется схемами декодирования адреса.

Первые БИС памяти имели логическую организацию вида N одноразрядных слов, или $N \times 1$, где N - количество адресов (одноразрядных слов) микросхемы. Следовательно, каждый разряд модуля памяти, построенного на таких микросхемах, включал свои собственные дешифраторы, буферные регистры и схемы управления, одного комплекта которых при традиционной организации было достаточно для целого модуля. Однако такое дублирование оправдывалось достигаемыми характеристиками памяти, а его стоимость не была чрезмерной (электроника обрамления составляла не более 5-15% от общей площади кристалла микросхемы).

Впоследствии разрядность хранимых в микросхеме слов была увеличена и составляет на сегодня от 4-х до 16-ти разрядов, т.е. *N* × 4, *N* × 8, *N* × 16. Понятно, что относительная доля избыточных схем обрамления при этом падает.

Сигналы управления БИС ОЗУ должны обеспечить возможность выбора следующих параметров:

1) конкретной ячейки памяти внутри ИС;
2) требуемой ИС (если в блоке их несколько);
3) направления обмена информацией между блоком ОЗУ и МП. (Операция ЧТЕНИЕ из ОЗУ в МП, ЗАПИСЬ – из МП в ОЗУ).

Входы управления направлением обмена могут быть реализованы в виде:

– двух отдельных сигналов W и R (рис. 2, а). В этом случае для реализации режима ЗАПИСЬ или ЧТЕНИЕ активизируются соответственно сигналы W или R. При этом активные значения этих сигналов (0 или 1) в различных ИС могут отличаться;

– совмещенного входа $W / \overline{R}$ или $\overline{W} / R$ (рис. 2, б, в). Этот вход используется следующим образом. При подаче на этот вход 1 – режим записи; 0 – режим чтения (или наоборот). Конкретное соединение этого входа с шиной управления представлено на рис. 3, б, в. При этом необходимо подсоединить к управляющему входу только один из сигналов шины управления (MEMR или MEMW) в зависимости от того, какой сигнал требуется. Это зависит от активного уровня сигнала ШУ (для МП КР580 – это 0) и активного уровня для входа управления БИС.

Входы и выходы данных могут быть реализованы следующим образом:

– Входы и выходы разделены (рис. 2, а, б). В этом случае необходимо согласовать две однонаправленные шины DI и DO с двунаправленной шиной ШД. Это создает определенные сложности, которые, однако, легко преодолимы.

– Входы/выходы данных двунаправленные (рис. 2, в). Такой вариант реализации предпочтительнее, так как позволяет упростить схему блока ОЗУ за счет исключения шинных формирователей. Однако при построении блока ОЗУ с использованием значительного числа ИС памяти все равно возникают проблемы нагрузочной способности шин, что может также потребовать дополнительных шинных формирователей для усиления сигналов в шинах.

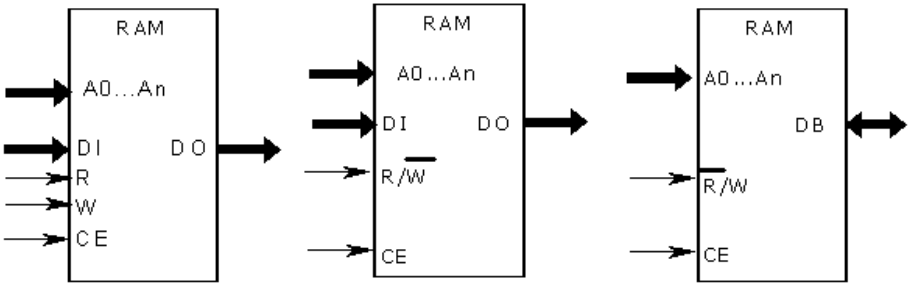


Рис. 2. Варианты реализации ИС ОЗУ

а) с раздельными входами и выходами данных и раздельными сигналами R, W, б) с раздельными входами и выходами данных и совмещенными сигналами R, W, в) с двунаправленными выходами и выходами данных и сигналами R, W.

Любое ЗУ собирается из интегральных схем. Организацию ЗУ можно представить в виде произведения (M x N) M - число ячеек памяти, N - разрядность одной ячейки памяти (рис. 3). Обычно M принимает одно из следующих значений 256, 512, 1024, 2048 ... . Число ячеек памяти в конкретной микросхеме можно определить, подсчитав число адресных входов а ИС. Тогда $M = 2^a$.

N обычно принимает следующие значения: 1, 2, 4, 8, 16, 32... .

Количество входов данных (*DI - Data Input*) равно разрядности хранимых слов. Количество выходов данных (*DO - Data Output*) также равно разрядности хранимых слов. Однако во многих случаях входы и выходы данных объединяются, что позволяет уменьшить вдвое количество выводов данных у микросхем памяти, а также упростить их подключение к шинам данных.

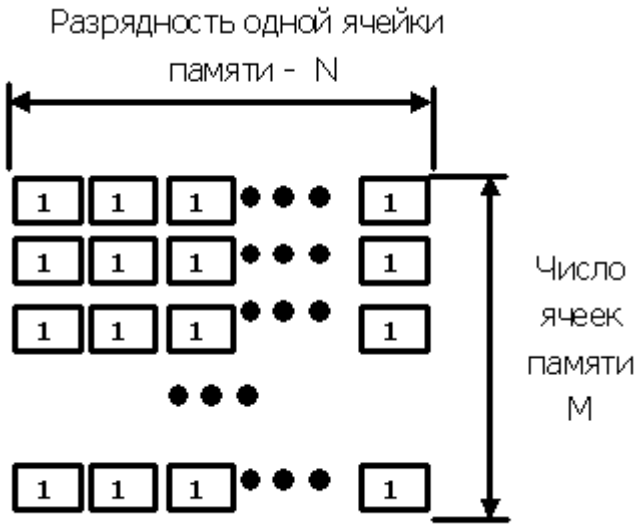


Рис. 3. Структура памяти

Общая емкость памяти (число бит информации, хранимой в памяти, или число триггеров для статического ОЗУ) V= M*N. Тогда микросхемы памяти следующей организации: 256x8, 512x4, 1024x2, 2048x1 –имеют одинаковую информационную емкость, но различную организацию.

Предположим, нам необходимо собрать блок ЗУ емкостью 2048x8 (2 кб). Рассмотрим варианты реализации этого блока на ИС ЗУ различной организации.

Вначале определим общее число ИС, требуемых для реализации данного блока.

При использовании ИС памяти различной организации (MxN) расчеты будут иметь следующий вид:

– при разрядности одной ячейки памяти ИС равной разрядности требуемого блока памяти (в нашем случае это ИС 256x8) делим число ячеек памяти в разрабатываемом блоке памяти P на число ячеек памяти в одной ИС. Тогда K = P / M = 2048 / 256 = 8.

– при разрядности одной ячейки памяти ИС, меньшей разрядности требуемого блока памяти (например, 1024x2), Реализуется следующий алгоритм:

1) делим разрядность ячейки памяти ИС на требуемую разрядность блока K1 = 8 / 2 = 4;

2) делим число ячеек памяти в разрабатываемом блоке памяти P на число ячеек памяти в одной ИС. Тогда K2=P/M = 2048/1024 = 2;

3) определяем общее число ИС, требуемых для реализации заданного блока, K = K1 * K2 = 4 * 2 = 8.

Так как суммарная емкость всех рассматриваемых ИС одинакова, то и результаты, полученные для двух типов ИС ЗУ, совпадают.

Реализация блока ОЗУ емкостью 2 кб на базе БИС ОЗУ 256x8 изображена на рис. 4.

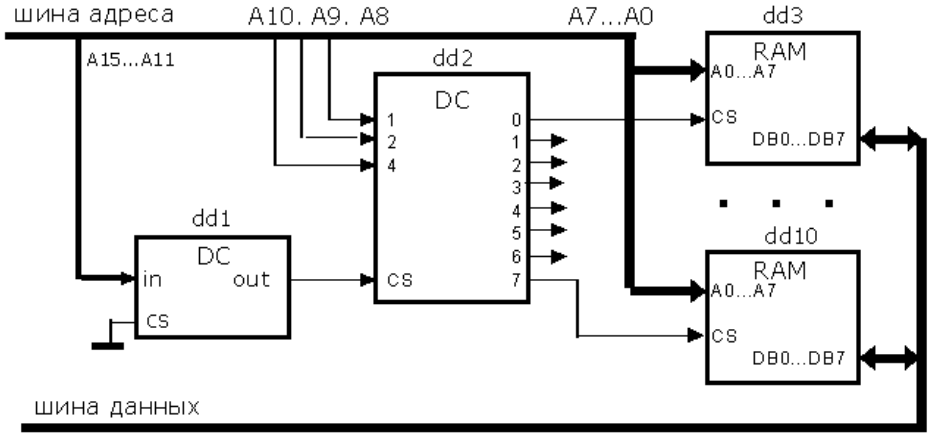


Рис. 4. Блок ОЗУ 2 кб. Вариант 1

Реализация блока ОЗУ емкостью 2 кб на базе БИС ОЗУ 2048x1 изображена на рис. 5.

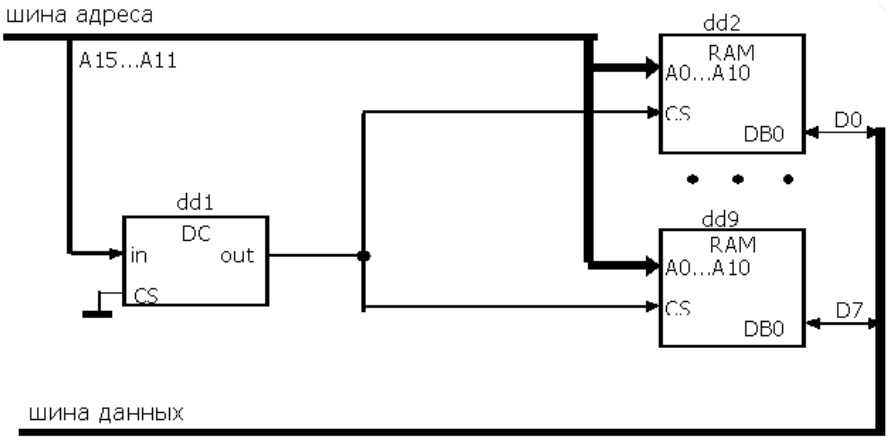


Рис. 5. Блок ОЗУ 2 кб. Вариант 2

Дешифратор для управления блоком ОЗУ должен обеспечивать решение следующих задач:

– задание положения блока ОЗУ в адресном пространстве МПС;

– выбор одной или группы ИС, если блок ОЗУ должен иметь больше ячеек памяти, чем имеется в одной ИС, из которых данный блок строится.

Соответственно формируются значения разрядов (A15 – A0) ША:

– A15 ... Aj+1 – старшая часть адреса задает положение блока в адресном пространстве МПС:

– Aj... Ai+1 – средняя часть разрядов ША используется для выбора одной из группы ИС (если число ячеек блока больше, чем в ИС)

– Ai... A0 – младшие разряды шины адреса служат для выбора ячейки памяти внутри ИС.

Пример 1. Рассмотрим реализацию блока памяти объемом 4 кб на ИС памяти 1024x2. Число ИС для реализации такого блока вне зависимости от их организации составляет 16 шт. Пусть начальный адрес блока памяти будет равен 7000H. Тогда в адресном пространстве МП КР580 этот блок будет расположен по адресам 7000H – 7FFFH. В этом случае каждая микросхема имеет 1024 ячеек и требует 10 линий, то есть младшие разряды шины адреса A9 – A0. Для выбора одной из группы ИС требуется 2 разряда ША – A11 – A10.

Для первого субблока он равен 00В, для второго 01В, для третьего 10В и для четвертого – 11В. Для выбора субблока можно использовать четырехразрядный дешифратор. Старшая часть адреса A15-A12 = 0111В и используется для задания положения блока в адресном пространстве МПС.

Таблица 1. Характеристики субблоков памяти

№	ИС субблока	Начальный адрес	Значения разрядов A15...A0 памяти для младшей ячейки	Значения разрядов A15...A0 памяти для старшей ячейки
1	DD01-DD04	7000H	7000H 0111 0000 0000 0000B	73FFH 0111 0011 1111 1111B
2	DD05-DD08	7400H	7400H 0111 0100 0000 0000B	77FFH 0111 0111 1111 1111B
3	DD09-DD12	7800H	7800H 0111 1000 0000 0000B	7BFFH 0111 1011 1111 1111B
4	DD13-DD16	7C00H	7C00H 0111 1100 0000 0000B	7FFFH 0111 1111 1111 1111B

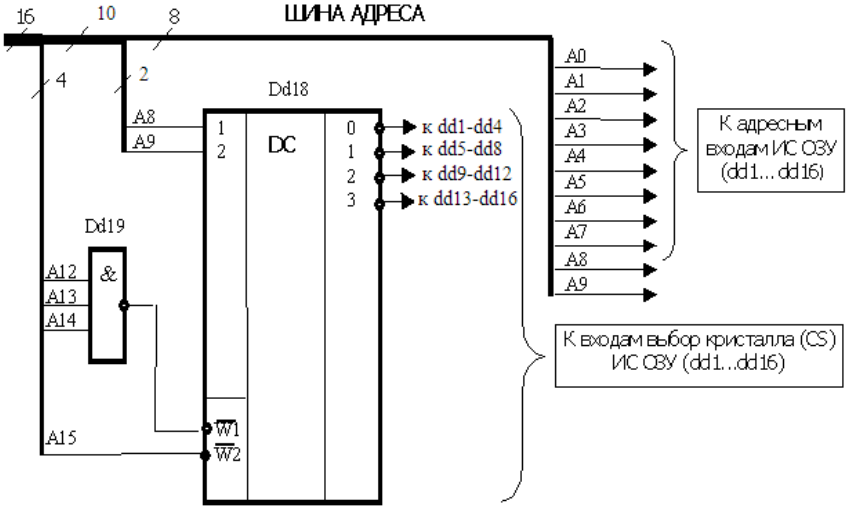


Рис. 6. Распределение сигналов шины адреса при построении блока ОЗУ емкостью 4 кб с начальным адресом 7000H на базе ИС памяти (1024x2)

Сигналы управления

Адресация — только часть процесса управления памятью и ВУ. Кроме адресов требуются стробы чтения и записи ($\overline{RD}$ и $\overline{WR}$), задающие направление обмена, сигналы разрешения работы ($\overline{CS}$, EN), признак обращения к ВУ или памяти (IO/M). Процессор обычно вырабатывает минимальную группу сигналов, тогда как в системном интерфейсе может быть предусмотрена несколько иная группа. В частности, МП К1821ВМ85А дает три сигнала: сигнал чтения ($\overline{RD}$), записи ($\overline{WR}$) и сигнал IO/M, т. е. обращения к ВУ при высоком уровне и к памяти — при низком. В системном же интерфейсе используется система из четырех сигналов: сигнала чтения из памяти $\overline{\text{MEMR}}$, записи в память $\overline{\text{MEMW}}$, чтения из ВУ $\overline{\text{IOW}}$ и записи в ВУ $\overline{\text{IOR}}$. К четверке сигналов легко перейти по следующим соотношениям:

$$\overline{\text{MEMR}} = \overline{\text{RD} \cdot \overline{\text{IO/M}}} = \overline{\text{RD}} \vee \text{IO/M};$$

$$\overline{\text{MEMW}} = \overline{\text{WR} \cdot \overline{\text{IO/M}}} = \overline{\text{WR}} \vee \text{IO/M};$$

$$\overline{\text{IOR}} = \overline{\text{RD} \cdot \text{IO/M}} = \overline{\text{RD}} \vee \overline{\text{IO/M}};$$

$$\overline{\text{IOW}} = \overline{\text{WR} \cdot \text{IO/M}} = \overline{\text{WR}} \vee \overline{\text{IO/M}}.$$

На рис. 7 представлены варианты подключения ОЗУ к шине управления.

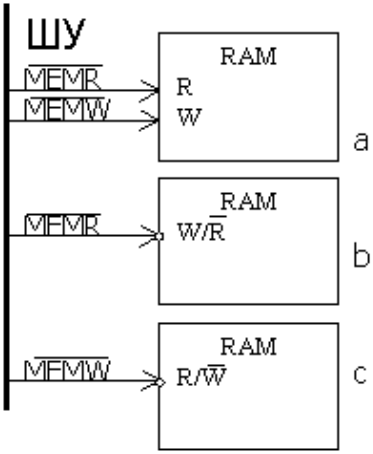


Рис. 7. Варианты подключения ОЗУ к шине управления

На рис. 8 изображена МПС с подключенными микросхемами ОЗУ.

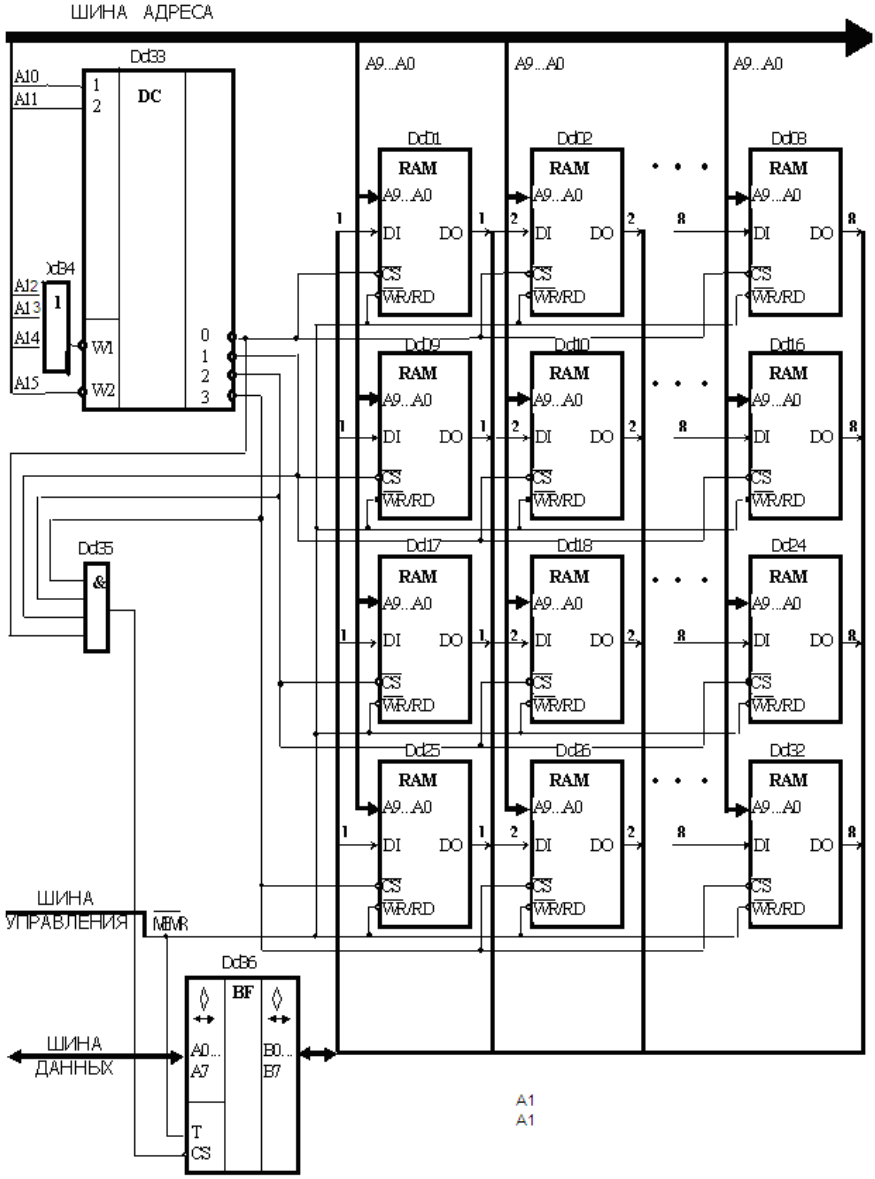
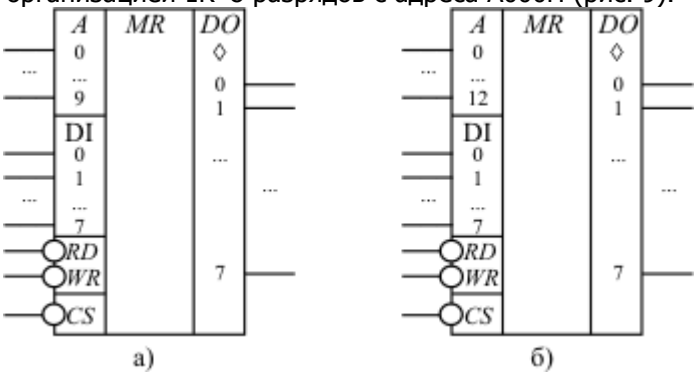


Рис. 8. Пример реализации блока ОЗУ емкостью 4 кб с начальным адресом 7000Н на базе микросхем ОЗУ 1024*1.

Пример 2. Построить ОЗУ с организацией 8К*8 разрядов на БИС с организацией 1К*8 разрядов с адреса А000Н (рис. 9).



а) б)

Рис. 9. Условно-графические обозначения запоминающих устройств с различной организацией: а) 1К*8 разрядов; б) 8К*8 разрядов

Модуль памяти будет состоять из восьми БИС. Для обращения к модулю памяти используется 16-разрядный адрес (А15 – А0), поступающий по шине адреса (ША). Три старших разряда А15-А13=101В=3Н определяют место памяти в адресном пространстве. Три средних разряда (А12-А10) определяют ту схему, которая в данный момент включается в работу, а каждая ячейка внутри любой БИС определяется 10-ю младшими разрядами адреса (А9-А0) (рис. 10).

Разряды адреса		Выбранная БИС
12 11 10 выбор БИС	9 … 0 выбор ячейки в БИС	
1 1 1	1…1 . . . 0…0	БИС 7
1 1 0	1…1 . . . 0…0	БИС 6
. . .		
0 0 1	1…1 . . . 0…0	БИС 1
0 0 0	1…1 . . . 0…0	БИС 0

Рис. 10. Организация модуля памяти

При единичном значении сигнала на входе выбора кристалла БИС (CS=1) выходные разряды данных находятся в третьем состоянии, то есть как бы отключены от шины (DO=Z).Таким образом, при любом значении кода на шине адреса всегда в работе находится одна и только одна из восьми БИС (рис. 11).

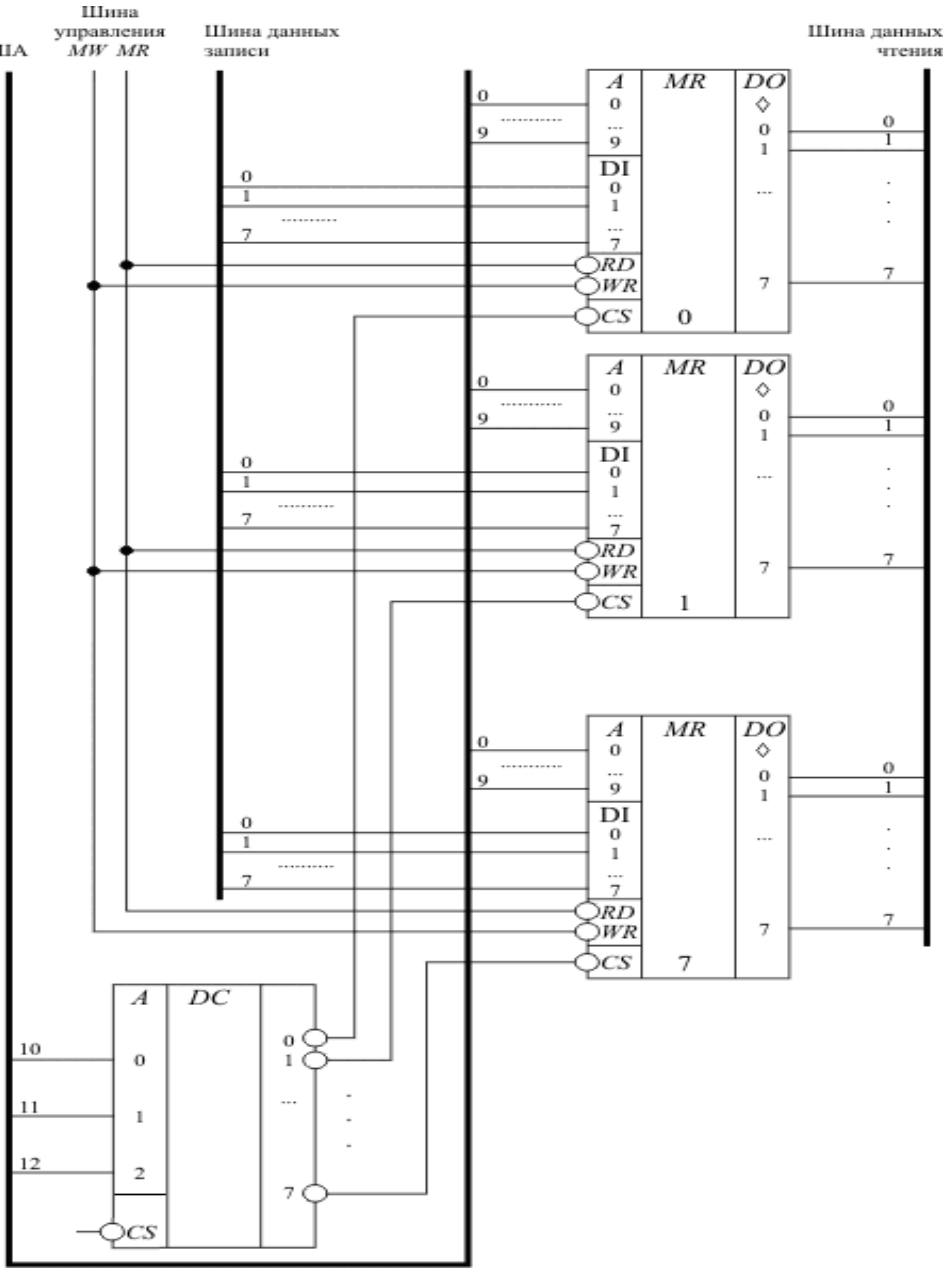

ША
Шина управления MW MR
Шина данных записи
Шина данных чтения
A
MR
DO
DI
RD
WR
CS
DC
10
11
12

Рис. 11. Запоминающее устройство объемом 8К*8 разрядов на БИС с организацией 1К*8 разрядов

На вход CS микросхемы DC на рис. 9 подать необходимо сигнал с микросхемы, изображенной на рис. 12.

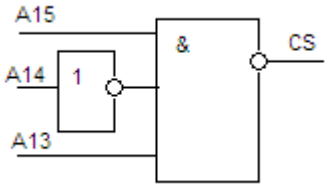


Рис. 12. Схема, задающая позицию ОЗУ в адресном пространстве МПС

Пример 3. Построить ОЗУ с организацией 1К*8 разрядов на БИС с организацией 1К*1 разряд (рис. 13).

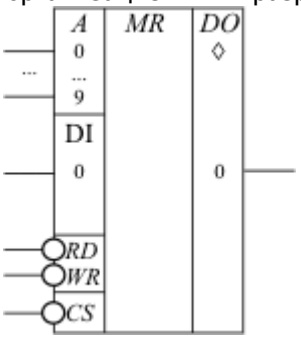


Рис. 13. Условно-графическое обозначение БИС с организацией 1К*1 разряд

В данном случае требуется увеличить разрядность слова памяти. Так как все разряды одного слова должны записываться и считываться одновременно, то все БИС должны работать параллельно. Модуль памяти будет состоять из восьми БИС (рис. 14). Если разрабатываемый блок является частью модуля памяти, имеющего объем больше, чем 1К слов (например, 8К), то необходим специальный дешифратор, который будет дешифрировать старшие разряды адреса аналогично тому, как показано на рис. 9 и включать в работу данный блок.

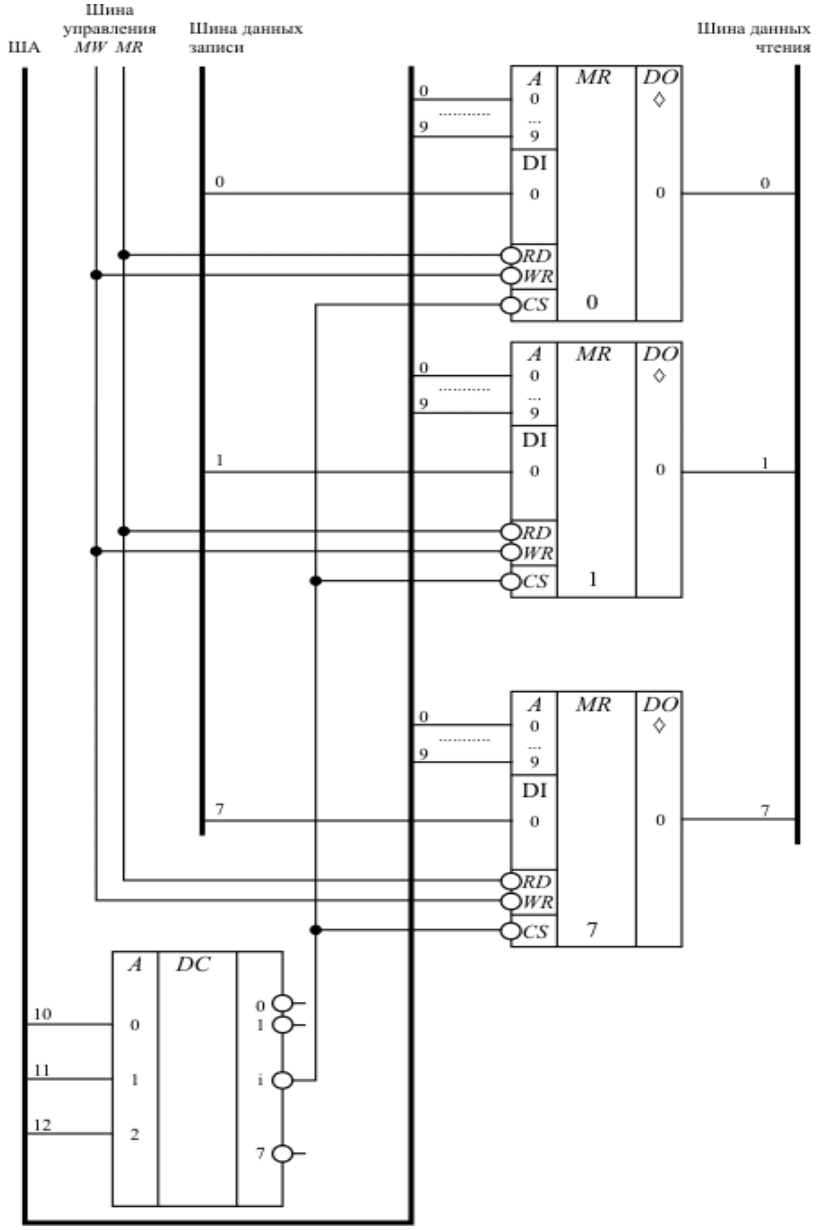


Рис. 14. Запоминающее устройство объемом 1К*8 разрядов на БИС с организацией 1К*1 разряд.

Другим общим сигналом, имеющимся почти во всех микросхемах, является сигнал выбора микросхемы - ***CS***# (*Chip Select*). Этот вход также обычно является инверсным и при единичном значении на нем микросхема переходит в "выключенное" состояние (выход данных микросхемы переходит в состояние высокого выходного сопротивления, если он является z-выходом, или в состояние "1", если это инверсный выход с открытым коллектором). При нулевом значении сигнала на входе ***CS***# микросхема находится в активном состоянии.

Постоянные запоминающие устройства

Очень часто в различных применениях требуется хранение информации, которая не изменяется в процессе эксплуатации устройства. Это такая информация как программы в микроконтроллерах или начальные загрузчики (BIOS) в компьютерах. В системе управления технологическими процессами может отсутствовать ОЗУ, а ПЗУ должно присутствовать всегда. Все ПЗУ (или часть) должны располагаться в адресном пространстве МП, начиная с адреса 0000H. Это связано с тем, что при формировании сигнала сброс центрального процессорного элемента или при включении питания программный счетчик PC устанавливается в состояние 0000H.

На принципиальных схемах ПЗУ обозначается как показано на рис. 13.

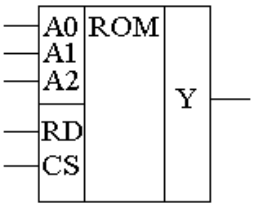


Рис. 13. Обозначение постоянного запоминающего устройства на принципиальных схемах

Для того чтобы увеличить разрядность ячейки памяти ПЗУ эти микросхемы можно соединять параллельно (выходы и записан-

ная информация естественно остаются независимыми). Схема параллельного соединения одноразрядных ПЗУ приведена на рис. 14.

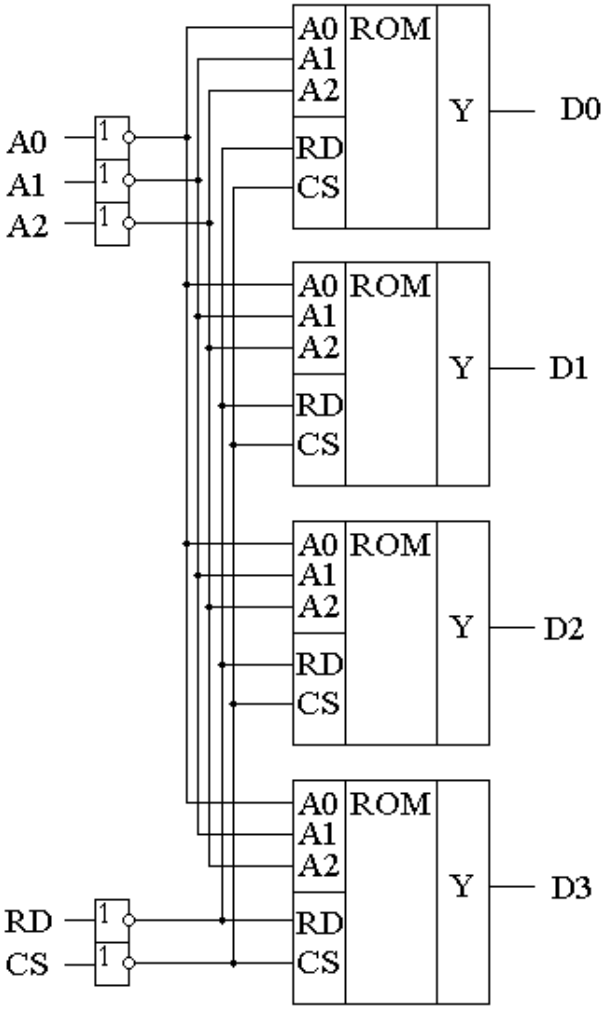


Рис. 14. Схема многоразрядного ПЗУ (ROM)

Основной особенностью подключения ПЗУ является необходимость обеспечить блокировку от записи по адресам памяти, занимаемым ПЗУ. Для реализации такой блокировки используются сигналы шины управления MEMR, MEMW. Конкретная схема зависит от используемой ИС ПЗУ. Для блокировки удобно использовать дополнительные входы CS ИС ПЗУ. Блокировка обеспечивает выбор соответствующей ИС не только при ее "правильном" адресе, но и при реализации операции чтения (MEMR). При подключении ИС памяти DD2 на рис. 15 блокировка реализована подачей сигнала MEMR на вход "выбор кристалла" дешифратора (W2) (DD1). Такой вариант блокировки удобно использовать при наличии свободного входа управления дешифратором и при единственном входе CS ИС ПЗУ.

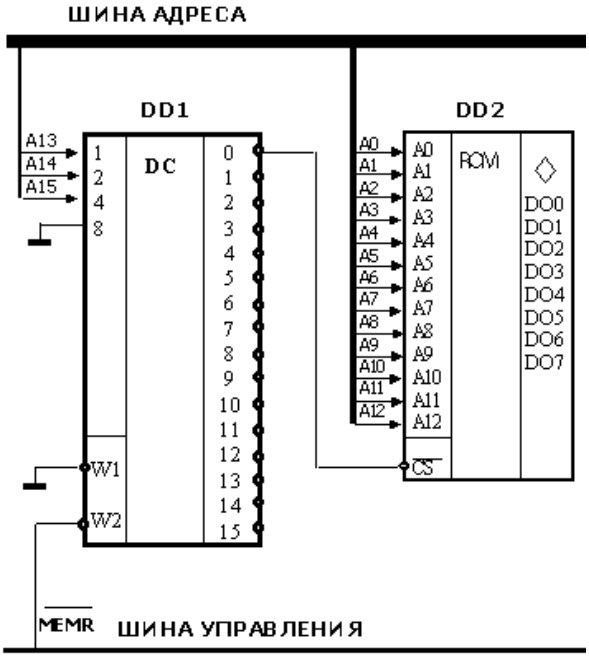


Рис. 15. Пример подключения ИС ПЗУ к шинам адреса и управления

Подключение устройств ввода/вывода.

Устройства ввода/вывода подключаются к тем же трем магистралям МПС - ША, ШД и ШУ. Правила подключения во многом похожи на рассмотренные выше правила подключения ОЗУ и ПЗУ.

Для устройств ввода/вывода существуют две возможности размещения в адресном пространстве:

– размещение в адресном пространстве памяти (обычно в зоне старших адресов). При таком способе подключения необходимо дешифрировать все 16 разрядов шины адреса и использовать сигналы управления обменом MEMW, MEMR. Число подключаемых устройств ограничивается только свободной зоной памяти системы;

– размещение в адресном пространстве ввода/вывода. В этом случае можно подключить до 256 устройств ввода и 256 устройств вывода. Для выделения конкретного устройства необходимо дешифрировать только восемь младших разрядов ША. Используются сигналы управления IOR, IOW.

Индивидуальные задания

Разработать способ подключения микросхем ОЗУ и ПЗУ с указанными характеристиками к микропроцессору, изображенному на рис. 16. При этом учесть необходимый объем памяти, требуемое адресное пространство.

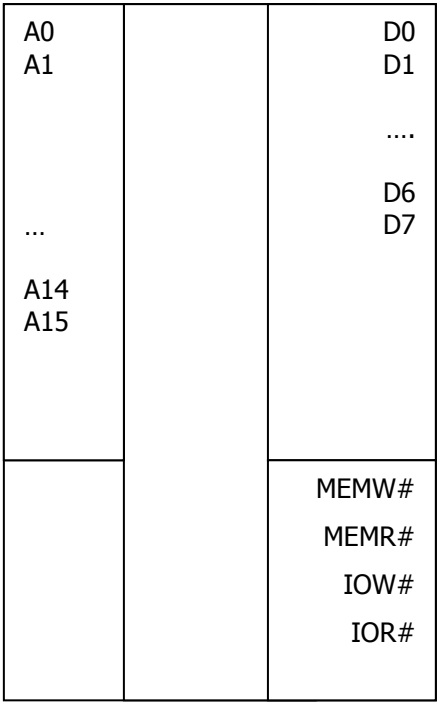


Рис. 16. Схема микропроцессора

Вариант 1. Микросхема ОЗУ имеет структуру 2 кБ*8. Микросхема ПЗУ имеет структуру 1 кБ*2. МПС должна содержать 8 кБ ОЗУ с ячейки $8000_{(16)}$ и 2 кБ ПЗУ с ячейки $0_{(16)}$.

Вариант 2. Микросхема ОЗУ имеет структуру 4 кБ*4. Микросхема ПЗУ имеет структуру 2 кБ*8. МПС должна содержать 16 кБ с ячейки $4000_{(16)}$ и 4 кБ ПЗУ с ячейки $0_{(16)}$.

Вариант 3. Микросхема ОЗУ имеет структуру 1 кБ*1. Микросхема ПЗУ имеет структуру 2 кБ*4. МПС должна содержать 8 кБ ОЗУ с ячейки $3000_{(16)}$ и 4 кБ ПЗУ с ячейки $3000_{(16)}$.

Вариант 4. Микросхема ОЗУ имеет структуру 8 кБ*4. Микросхема ПЗУ имеет структуру 1 кБ*8. МПС должна содержать 16 кБ с ячейки $2000_{(16)}$ и 4 кБ ПЗУ с ячейки $0_{(16)}$.

Контрольные вопросы

1. Что такое адресное пространство МПС?
2. Какие управляющие сигналы имеются у микросхем ОЗУ и ПЗУ?
3. Для чего служат сигнал чтения из памяти $\overline{\text{MEMR}}$, записи в память $\overline{\text{MEMW}}$, чтения из ВУ $\overline{\text{IOW}}$ и записи в ВУ $\overline{\text{IOR}}$?
4. Как определить количество линий шины адреса у микросхемы ОЗУ по известному количеству ячеек памяти в ней?
5. Как осуществляется выбор конкретной микросхемы памяти по номеру ячейки?

ЛАБОРАТОРНАЯ РАБОТА 8
АРХИТЕКТУРА И СИСТЕМА КОМАНД МИКРОПРОЦЕССОРА X86

Цель работы: ознакомиться с отладчиком Turbo Debugger, архитектурой микропроцессора и научиться использовать элементарные операции Ассемблера.

Краткие теоретические сведения

1. ЗНАКОМСТВО С ПРОГРАММОЙ-ОТЛАДЧИКОМ TURBO DEBUGGER. ВЫПОЛНЕНИЕ ПРОСТЕЙШИХ КОМАНД МИКРОПРОЦЕССОРА В СРЕДЕ TURBO DEBUGGER.

Понятие отладки. Назначение программ-отладчиков

Отладка (debugging) — один из важнейших этапов разработки программного обеспечения (английский термин bug означает "ошибка в программе"). В процессе отладки путем детального анализа в компьютерных программах выявляются и устраняются возможные логические ошибки, которые не обнаруживаются на стадии компиляции.

Отладчики (debugger) — это вспомогательные программы (утилиты), включаемые в набор инструментальных средств программиста для выполнения отладки других программ. Отладчики предоставляют программисту возможность выполнять программу по шагам, следить за изменениями данных и проверять выполнение условий. В зависимости от уровня языка, которым оперирует отладчик, можно выделить два их типа.

Отладчики исходного кода дают программисту возможность видеть текст программы на языке высокого уровня (например, Си), проверять значения отдельных переменных и агрегатов данных (таких, как массивы), используя их имена.

Отладчики машинного уровня отслеживают реально выполняемые машинные команды, отображаемые в виде команд ассемблера. Они позволяют также просматривать содержимое ячеек памяти и регистров микропроцессора.

Отладчик, интегрированный в среду разработки программ пакета Borland C++, относится к первому типу. Turbo Debugger — это отладчик второго типа.

Запуск программы

Запуск программы осуществляется файлом td.exe.

Структура экрана программы Turbo Debugger

При запуске Turbo Debugger на экране появляется его основное меню и рабочее окно рис.1.

Рабочее окно состоит из следующих четырёх окон:

1. окно команд – ***CPU***;
2. окно регистров и флагов – ***Registers***;
3. окно данных - ***Dump***;
4. окно стека.

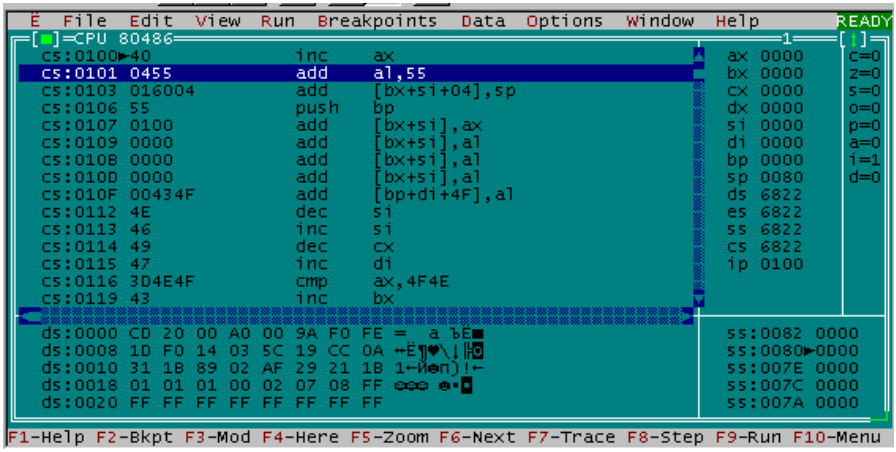


Рисунок 1

В свою очередь окно ***Registers*** поделено на две части. В левой его части указано содержимое всех регистров микропроцессора (ax,bx,cx,dx…), а в правой части показаны биты состояния (флаги-c,z,s,o….).

Последовательное переключение между окнами можно выполнять с помощью клавиши Tab (или Shift+Tab в обратном порядке).

Каждое из окон может быть вызвано самостоятельно на экран, используя пункт меню View и команду соответствующую названию окна (***CPU, Registers, Dump***).

Задание 1. *Выполнить переключение между окнами, используя разные приемы.*

Регистры микропроцессора

Регистры — это небольшие (несколько байт) именованные области памяти микропроцессора, используемые для временного хранения двоичных данных, к которым необходимо обеспечить быстрый доступ.

Каждый регистр может иметь специальное назначение, например, хранить операнды команд микропроцессора, хранить адрес очередной команды программы и т.п. В микропроцессорах Intel для регистров в целом и отдельных групп байт из них принята специальная система обозначений. Например, имеется группа двухбайтовых *регистров общего назначения*, обозначаемых AX, BX, CX и DX. Например, регистры AX и BX используются микропроцессором для извлечения значений операндов операций сложения и вычитания, а также размещения результатов выполнения соответствующих инструкций.

Как было описано выше, окно ***Registers*** поделено на две части. В левой части показано содержимое регистров. Обратим внимание на первые четыре регистра AX, BX, CX, DX, они все равны 0000, четырехзначное число, показанное вслед за именем регистра, является шестнадцатеричным.

Для организации вычислений микропроцессор i8086 имеет в своём составе 14 шестнадцатиразрядных регистров, которые обеспечивают выполнение программы:

Регистры общего назначения			Сегментные регистры	Специальные регистры	
AH	AL	AX	CS	SP	Указатель стека
BH	BL	BX	DS	BP	Указатель базы стека
CH	CL	CX	ES	IP	Указатель инструкций
DH	DL	DX	SS	FLAGS	Регистр флагов
SI					
DI					

Регистры общего назначения:

AX(AH, AL), BX(BH, BL), CX(CH, CL), DX(DH, DL) делятся программно на пары однобайтных регистров и могут использоваться для хранения данных. Разбиение на однобайтные регистры позволяет увеличить общее число регистров;

SP, BP – указатель и база стека, соответственно, обеспечивают доступ к данным в стеке, могут использоваться для хранения данных, но делать это не рекомендуется, так как при этом возможно нарушение адресации в стеке, особенно при использовании SP.

SI, DI – шестнадцатиразрядные регистры для хранения данных.

CS, DS, ES, SS – хранят адреса сегментов в памяти, не могут использоваться для хранения данных.

IP – регистр инструкций – хранит адрес (смещение) следующей исполняемой команды.

FLAGS – регистр флагов содержит набор битовых флагов, определяющий текущее состояние процессора и результат выполнения предыдущей команды (таблица 1).

Таблица 1

Регистр флагов процессора		
Флаг	Название	Назначение
O	Переполнение	Переполнение при выполнении арифметических операций
D	Направление	Направление пересылки данных при выполнении строковых команд
I	Прерывание	Разрешает/Запрещает внешние прерывания
T	Пошаговый режим	Останов после выполнения каждой команды(используется отладчиками)
S	Знак	Знак результата выполненной команды(0 – плюс, 1 – минус)
Z	Ноль	Значение результата выполненной команды(0 – ненулевой, 1 – нулевой)
A	Внешний перенос	Используется для специальных арифметических операций
P	Контроль чётности	Число единиц в операнде(0 – нечётное, 1 – нечётное)
C	Перенос	Содержит перенос из старшего бита при выполнении арифметических операциях

Сложение беззнаковых величин

Под беззнаковым понимается представление заведомо неотрицательных целых чисел, в котором знаковый бит вводить не требуется.

Пусть необходимо сложить числа 3A7h и 92Ah, записанные в шестнадцатеричной форме (подсчитайте результат).

Создадим и выполним инструкцию микропроцессора, позволяющую складывать беззнаковые коды из двух регистров — прибавим число, хранящееся в регистре BX, к числу хранящемуся в AX . Для этого необходимо выполнить следующую последовательность действий.

• Занесение операндов.

Поместим заданные числа в регистры AX, BX. Для того чтобы изменить содержимое регистра (занести величину), необходимо установить на него курсор в левой части окна ***Registers*** и ввести нужное число.

• Создание инструкции.

Сложение чисел будет производиться при выполнении специальной команды микропроцессора. Чтобы команду можно было выполнить, ее нужно сохранить в основной оперативной памяти компьютера. Для этого вначале нужно определить адрес, где она будет храниться, и только потом ввести саму команду.

Для указания адреса необходимо переместится в окно команд (клавиша Tab).

а) Ввод адреса.

Разместим нашу инструкцию по адресу 100h (с этого адреса отладчик размещает первый байт инструкции, в любом сегменте памяти, который он начал использовать). Если в данный момент курсор находится по другому адресу, то для его перемещения в нужное место нажмите сочетание клавиш Ctrl+G и в диалоговом окне укажите нужный адрес, рис 2.

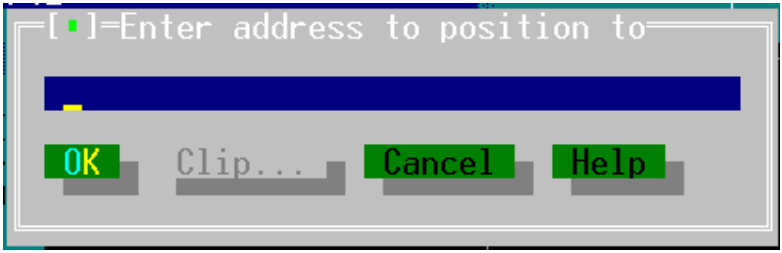


Рисунок 2

б) Ввод команды.

Сложение чисел задается ассемблерной командой ADD (английское add означает "сложить" или "добавить").

Поэтому занесите по выбранному адресу инструкцию

ADD AX, BX

в) Подготовка к выполнению команды.

Чтобы выполнить команду, необходимо сообщить микропроцессору адрес нашей инструкции. Переместимся в окно регистров и флагов ***Registers***. Адрес инструкции заносится либо непосредственно в регистр IP (instruction pointer — указатель команды), либо в диалоговое окно при нажатии сочетания клавиш Ctrl+N. При этом в строчке между адресом инструкции и кодом появляется метка-треугольник, обозначающая инструкцию, которая будет выполнена процессором следующей.

г) Выполнение команды.

Для выполнения инструкции выберем команду Trace into меню Run или нажмем клавишу F7. Команда Trace into выполняет одну инструкцию за шаг.

д) Результат выполнения команды.

После выполнения команды ADD результат вычисления помещается в регистр AX, где ранее был один из операндов. Поэтому после сложения регистр AX должен содержать число CD1h.

Задание 2. Выполните сложение следующих шестнадцатеричных чисел

1.	1C6+223=?	2.	192+258=?	3.	29E+14=?	4.	28F+15D=?
5.	1DF+20E=?	6.	2AA+144=?	7.	1BB+234=?	8.	1CC+224=?
9.	1FF+1F2=?	10.	1EE+204=?	11.	1AB+248=?	12.	1BA+23A=?
13.	1AC+249=?	14.	1CA+22C=?	5.	1AD+24A=?	16.	1DA+21E=?

Вычитание беззнаковых величин

Вычитание выполняется с помощью команды SUB (subtract — вычесть). В остальном все этапы выполнения вычисления повторяют действия, которые были описаны для операции сложения. В регистр AX заносится уменьшаемое, а в регистр BX — вычитаемое. Результат выполнения инструкции появится в регистре AX.

Задание 3. Проверим результат, полученный при выполнении сложения. Вычтем из числа CD1h одно из ранее использованных слагаемых (для определенности 92Ah). Каков результат операции?

Если в выражении вычитаемое больше уменьшаемого, результат вычитания беззнаковых (неотрицательных) величин становится отрицательным.

Задание 4. Проверим, как микропроцессор использует форму двоичного дополнения для представления отрицательных результатов. Выполните вычитание из нуля единицы (т.е., 0 – 1). Какой результат получен и почему?

Существует еще одна пара команд увеличения и уменьшения на 1 – Inc (increment) и Dec (decrement). Команда INC аналогична команде:

ADD op,1

т.е. увеличивает свой операнд на 1: op1:=op1+1, а команда DEC аналогична команде:

SUB op,1

т.е. уменьшает операнд на 1: op1:=op1-1(единственное отличие: команды INC и DEC не меняют флаг переноса CF).Например, если регистр BX содержал число 0001, то после выполнения команды Inc BX регистр будет содержать 0002. Выгода от команд INC и DEC в том, что они занимают меньше места в памяти и выполняются быстрее, чем соответствующие команды ADD и SUB.

Операции с байтами

В микропроцессорах Intel используются двухбайтовые машинные слова. Каждый регистр общего назначения (AX, BX, CX и DX) может хранить одно машинное слово. Однако имеется возможность оперировать с отдельными байтами этих регистров. В этом случае каждый регистр рассматривается состоящим из старшего (High) и младшего (Low) байтов. Обозначения отдельных байтов из регистров состоят из двух букв. Первая задает имя регистра (A, B, C или D), а вто-

рая указывает, какой это байт регистра. Для обозначения старшего байта используется буква H, а младшего — L. Таким образом, регистр AX можно рассматривать, состоящим *из двух однобайтовых регистров* AH и AL.

Микропроцессор может выполнять арифметические операции над отдельными байтами.

Задание 5.

Введите в регистр AX число 0102h (два байта) и выполните инструкцию

ADD AH, AL.

Каков результат выполнения операции, который будет помещен в регистр AH?

Умножение беззнаковых величин

Умножение двух 16-битных чисел может дать 32-разрядный результат, поэтому инструкция умножения MUL (multiply — умножить) размещает результат в двух регистрах DX и AX. Старшие 16 бит помещаются в регистр DX, а младшие в AX.

Замечание.

При выполнении операции умножения одним из множителей всегда является значение из регистра AX.

Задание 6.

Выполните умножение чисел 7C4Bh (в регистр AX) и 100h (BX).

Каков результат операции и почему?

Деление беззнаковых величин

Команды микропроцессора предназначены для выполнения целочисленных операций. Так как деление целых чисел нацело происходит далеко не всегда, то результат деления формируется из двух целых чисел — частного и остатка от деления.

Делимое всегда помещается в пару регистров AX, DX, поэтому в инструкции деления DIV (divide — делить) необходимо указать только регистр с делителем. После выполнения деления регистр AX будет содержать **частное**, а регистр DX — **остаток**.

Задание 7.

Выполните деление числа 7C4B12h (DX=007Ch, AX=4B12h) на 0100h (BX).

Каков результат выполнения операции и почему?

Пересылка (копирование) данных

Для изменения содержимого регистров программным путем обычно используют команду MOV (move — внести), которая позволяет копировать в один регистр число или содержимое другого регистра. Первый операнд инструкции MOV указывает адресат (куда переслать значение), а второй — пересылаемое значение или регистр, его содержащий.

Задание 8.

С помощью команды MOV поместите числа 1234h и ABCDh, соответственно, в регистры AX и DX. Соответствующие инструкции поместите по адресам 0100h и 0102h.

Далее с помощью команды MOV поместите содержимое младшего байта регистра DX в старший байт регистра AX.

Понятие переполнения

Как и в случае умножения, при выполнении сложения результат может выходить за 16-разрядную сетку (четыре шестнадцатеричных числа). Например, результатом сложения четырехзначных чисел FFFFh и 1h будет пятизначное число 10000h, для записи которого слова (двух байт) недостаточно.

Если результат выполнения операции (над беззнаковыми величинами!) не может быть полностью размещен в регистре, то говорят о возникновении **переполнения**.

При выполнении сложения беззнаковых чисел суть переполнения (в двоичном представлении) состоит в том, что в результате сложения двух единиц в старшем разряде возникает единица, выходящая за разрядную сетку результирующего регистра. Естественно, что эта единица в регистр помещена быть не может, и при записи в регистр отсекается.

Задание 9.

Выполните сложение чисел FFFFh (AX) и 1h (BX). Каков результат операции?

Регистр флагов.

Флаг - это бит, принимающий значение 1 ("флаг установлен"), если выполнено некоторое условие, и значение 0 ("флаг сброшен") в противном случае. В ПК используется 9 флагов, причем конструктивно они собраны в один 16-разрядный регистр, называемый регистром флагов и обозначаемый как Flags. Эти биты обозначаются буквами *C,*

P, A, Z, S, T, I, D, O. Например, в текстовый редактор загружен текст. Как только вы внесли в текст первое изменение, можно установить в 1 флаг изменений. После сохранения текста значение флага сбрасывается (0). Тогда при выходе из редактора легко проверить, сохранены ли изменения.

Флаг переноса

Если при сложении беззнаковых чисел происходит переполнение (возникает единица переноса за пределы разрядной сетки регистра), то единичка переноса записывается в Carry Flag. В правой половине окна регистров и флагов (***Registers***) данный флаг обозначается буквой C. Флаг переноса переустанавливается в каждой операции сложения.

Задание 10.

Проследите за изменением состояния флага переноса при последовательном выполнении следующих операций

1. FFFF + 1
2. FF00 + 1

Использование флага переноса

I. Сложение с использованием флага переноса.

Рассмотренная ранее инструкция сложения ADD выполняет простое сложение двух беззнаковых кодов. Инструкция ADC складывает три числа: два операнда из регистров общего назначения, как и раньше, плюс значение бита флага переноса из регистра флагов.

Задание 11.

а) Выполните сложение FFFFh и 1.

б) Затем выполните инструкцию: ADC BX, AX

В результате сложения 1 и 0, в регистре BX будет число 2. (надо пояснить, что останется в регистрах после первой операции).

II. Вычитание с использованием флага переноса.

При выполнении инструкции SBB из разности операндов вычитается значение флага переноса.

Задание 12.

а) Выполните сложение FFFFh и 1.

б) Затем выполните инструкцию: SBB BX, AX

Флаг нуля.

Занесите в регистры BX и AX два равных числа, теперь инструкцией SUB произведите вычитание одного числа из другого, в результате чего должен быть установлен флаг нуля Z=1 (Zero Flag).

Флаг знака.

Данный флаг позволяет узнать знак числа. Если вычесть из нуля единицу, то результат будет FFFFh, при этом устанавливается флаг знака S=1 (Sign Flag).

Флаг переполнения.

Флаг переполнения устанавливается в той ситуации, когда этого не должно было произойти. Занесите в регистр AX число 7000h, а в BX 6000h и выполните инструкцию сложения, в результате AX будет содержать число D000h или-12288. Это ошибка, так как результат переполняет слово и является отрицательным, поэтому микропроцессор устанавливает флаг переполнения O=1(Overflow Flag)

Структура памяти

Память, с которой взаимодействует процессор при обработке программ, называется Оперативным Запоминающим Устройством (ОЗУ) или Random Access Memory (RAM). Она состоит из набора однобайтных ячеек, обращение к которым происходит по их номерам (физическим адресам). Число ячеек зависит от ширины шины адреса и составляет для процессора i8086 (ширина шины адреса равна 20) 2^{20} – ячеек (1Мбайт). Для современных процессоров с шириной шины адреса 32 объём ОЗУ может доходить до 4 Гбайт.

Данные можно читать или сохранять в ОЗУ байтами, указывая номер требуемой ячейки или словами (2 байта), указывая адрес младшей ячейки памяти и вводя специальный префикс.

Сегментация памяти

Для обращения к памяти процессор предварительно помещает адрес ячейки в один из своих регистров, но для процессора i8086, очевидно нельзя в шестнадцатиразрядном регистре хранить двадцатиразрядный адрес. Поэтому применяют так называемую сегментацию памяти, которая заключается в том, что истинный, физический адрес ячейки хранится в двух регистрах.

Один из них – сегментный, он хранит адрес начала блока памяти, который и называется сегментом. Если к шестнадцати разрядам сегмента мысленно справа дописать четыре двоичных нуля(16+4=20),

то получим физический адрес начала сегмента в ОЗУ. Второй регистр хранит величину смещения адреса требуемой ячейки от начала сегмента. Адрес ячейки памяти записывается в виде двойного слова (4 байта): <сегмент>:<смещение>.

Сегмент всегда начинается с ячейки, номер которой заканчивается на 4 двоичных (или один шестнадцатеричный) нуля. Минимальная длина сегмента 16 байтов (параграф). Максимальная длина определяется длиной регистра, хранящего смещение и равна 2^{16}(64 Кбайта).

Пара регистров CS:IP(<сегмент>:<смещение>) определяют адрес следующей команды программы.

Для адресации данных используются сегментные регистры DS и ES, а в качестве регистров, хранящих смещение, используются регистры общего назначения BX, SI, DI. Для работы с сегментом стека используют сегментный регистр SS и регистр BP.

Режимы адресации

1. Регистровая прямая - операнд находится в регистре.

Обозначение - <регистр>,

< регистр > - AX, BX, CX, DX, SI, DI, BP, SP, AL, BL, CL, DL, AH, BH, CH, DH.

Пример:

mov AX,SI ; переслать содержимое регистра SI в регистр AX.

2. Непосредственная - непосредственный операнд (константа) присутствует в команде.

Обозначение - < константное выражение > .

Пример:

mov AX, 093Ah ; занести константу 093Ah в регистр AX.

3. Прямая - исполнительный адрес операнда присутствует в команде.

Обозначение - < переменная >+/-< константное выражение >.

Пример:

mov AX, 0x100; переслать в AX слово памяти с именем WW

mov BX, 0x100+2 ; переслать в BX слово памяти отстоящее от переменной с именем WW на 2 байта.

4. Регистровая косвенная - регистр содержит адрес операнда.

Обозначение - [< регистр >],

< регистр > - BX. BP. SI, DI.

Пример:

mov [BX], CL ; переслать содержимое регистра CL по адресу, находящемуся в регистре BX.

5. Регистровая относительная - адрес операнда вычисляется как сумма содержимого регистра и смещения.

Обозначение - < переменная >[< регистр >] или [< регистр >]< константное выражение >,

< регистр > - SI или DI индексная адресация, BX или BP - базовая адресация.

Пример:

mov AX, WW[SI] ; переслать в AX слово из памяти, адрес которого вычисляется как сумма содержимого регистра SI и смещения WW.

6. Индексно - базовая - адрес операнда вычисляется как сумма содержимых базового и индексного регистров и смещения.

Обозначение - [< базов. регистр>][< индексн. регистр>] или <переменная >[<базов. регистр >][< индекс. регистр >] или [<базов. регистр >][< индекс. регистр >]< константное выражение,

где < индекс. регистр > - SI или DI, < базов. Регистр > - BX или BP.

Пример:

mov [BX+ SI+ 2], CL; переслать содержимое регистра CL по адресу, вычисляемому как сумма содержимого регистров BX, SI и константы 2.

Инструкции пересылки данных и двоичной арифметики

Команды данной группы приведены в таблице 2. Код определяет выполняемое командой действие, операнды показывают адреса ячеек, хранящих исходные данные, необходимые для выполнения команды и адрес ячейки результата. Процессор i8086 и более поздние версии относятся к двухадресным машинам. Это значит, что его команда может содержать не более двух операндов. Если для выполнения команды необходимо иметь два источника данных, например, сложение, то сохранение результата выполнения команды производиться по адресу одного из источников данных. Чтобы показать, какой из операндов будет хранить результат, его обозначают при описании команды как dst (destination - назначение), операнд, который исполь-

зуется только как адрес исходных данных, обозначается как src (source – источник). В двухоперандных командах операнд dst указывает, перед выполнением команды, адрес исходного данного, а после выполнения - адрес результата.

Таблица 2

Команды пересылки и двоичной арифметики

Мнемокод		Флаги						*Действие*
Код	Операнды	O	S	Z	A	P	C	
mov	dst, src.	-	-	-	-	-	-	пересылка
xchg	dst, src	-	-	-	-	-	-	обмен
add	dst, src	x	x	x	x	x	x	сложение
adc	dst, src	x	x	x	x	x	x	сложение с переносом
inc	dst	x	x	x	x	x	-	увеличить на единицу
sub	dst, src	x	x	x	x	x	x	вычитание
sbb	dst, src	x	x	x	x	x	x	вычитание с заемом
dec	dst	x	x	x	x	x	-	уменьшение на единицу
neg	dst	x	x	x	x	x	x	изменение знака
rcl	dst,счетчик	x	-	-	-	-	x	циклический сдвиг влево
rcr	dst,счетчик	x	-	-	-	-	x	циклический сдвиг вправо
rol	dst,счетчик	x	-	-	-	-	x	циклический сдвиг влево
ror	dst,счетчик	x	-	-	-	-	x	циклический сдвиг вправо
sal	dst,счетчик	x	x	x	u	x	x	арифметический сдвиг влево
sar	dst,счетчик	x	x	x	u	x	x	арифметический сдвиг вправо
shl	dst,счетчик	x	x	x	u	x	x	логический сдвиг влево
shp	dst,счетчик	x	x	x	u	x	x	логический сдвиг вправо
push	src	-	-	-	-	-	-	сохранение слова в стеке
pop	Dst	-	-	-	-	-	-	восстановление слова из стека
xlat	таблица	-	-	-	-	-	-	трансляция байтов из таблицы
lea	dst, src	-	-	-	-	-	-	загрузка исполнительного адреса
lds	dst, src	-	-	-	-	-	-	загрузка указателя с DS
les	dst, src	-	-	-	-	-	-	загрузка указателя с ES
lahf		-	-	-	-	-	-	загрузка флагов в AH
sahf		-	r	r	r	r	r	установка флагов из AH
pushf		x	-	-	-	-	x	сохранение флагов в стеке
popf		r	r	r	r	r	r	восстановление флагов из стека

Примечание:

- Флажок не модифицируется

x Устанавливается или сбрасывается в соответствии с результатом;

u Не определен;
r Восстанавливается прежнее запомненное значение.

Пример выполнения работы

Вычислить X = 3A + (B + 5) / 2 - C - 1,

где A, B, C, X- целые знаковые числа занимающие слово, написать программу реализующую данную формулу.

Распишем формулу по отдельным операциям:

```
AX ← A ; значение A в регистре AX
AX ← 2 *( AX )          ; 2A в AX
AX ← ( AX ) + A         ; 3A в AX
BX ← B                  ; B в BX
BX ← 5 + ( BX )         ; B+5 в BX
BX ← (BX ) / 2          ; ( B+5) / 2 в BX
AX ← (BX ) + ( AX )     ; 3A+( B+5 ) / 2 в AX
AX ← ( AX ) - C         ; 3A+( B+5 ) / 2 - C в AX
AX ← ( AX ) -1          ; 3A+( B+5 )/2 - C - 1 в AX
X ← ( AX )              ; 3A+( B+5 )/2 - C - 1 в X
```

Текст программы:

```
mov ax, a               ; a - адрес расположения переменной A, например 0x100
sal ax, 1
add ax, a
mov bx, b               ; b - адрес расположения переменной B, например 0x102
add bx, 5
sar bx, 1
add ax, bx
sub ax, c       ; c - адрес расположения переменной C, например 0x104
dec ax
mov x, ax               ; запись результата в память x, например x = 0x106.
```

Варианты заданий

Разработать программу, реализующую указанную формулу, исполнить программу с несколькими (три - четыре) наборами исходных данных, проверить правильность результатов.

1. X= A - 5 (B - 2C) + 2
2. X= - 4A + (B + C) / 4 + 2
3. X= 7A - 2B - 100 + C
4. X= - A / 2 + 4 (B + 1) + 3C
5. X= 5 (A - B) - 2C + 5
6. X= (A/ 2 + B) / 4 + C - 1
7. X= - (C + 2A + 4B + B)
8. X= 6C + (B - C + 1) / 2
9. X= 2 - B (A + B) + C / 4
10. X= 2B - 1 + 4 (A - 3C)
11. X= (2A + B) / 4 - C / 2 + 168
12. X= 6 (A - 2B + C / 4) + 10
13. X= 5 (A - B) + C mod 4
14. X= - (- (C + 2A) * 4B + 38)
15. X= A - 3 (A + B) + C mod 4
16. X= 3(A - 2B) +50 – C / 2
17. X= (3A + 2B) - C / 4 + 217
18. X= 3(C - 2A) + (B - C + 1) / 2
19. X= (2A + B) / 4 - C / 2 + 168
20. X= 6 (A - 2B + C / 4) + 10
21. X= 3 (A - 4B) + C / 4
22. X= - (- (C + 2A) * 5B - 27)
23. X= A / 2 - 3 (A + B) + C * 4
24. X= 3(A - 2B) +50 – C / 2
25. X= 5A + 2B - B / 4 + 131

Контрольные вопросы

1. Каковы задача и содержание этапа отладки программ?
2. Типы программ-отладчиков и особенности их работы.
3. Понятие регистра микропроцессора и машинного слова.
4. Какая инструкция позволяет выполнять сложение целых чисел? Где размещаются операнды и результат?
5. Какова последовательность выполнения инструкции сложения чисел в среде программы Turbo Debugger?
6. Какая инструкция позволяет выполнять вычитание целых чисел? Где размещаются операнды и результат?
7. В каком виде микропроцессор представляет отрицательные числа? Как будет представлен результат выполнения операции 5h – 8h?
8. Поясните особенности представления и именования двухбайтовых регистров общего назначения в виде совокупности двух однобайтовых.
9. Какими особенностями обладает инструкция умножения целых чисел? Где размещаются операнды и результат?
10. Какими особенностями обладает инструкция деления целых чисел? Где размещаются операнды и результат?
11. Поясните, что означает термин "переполнение". Как выяснить, что при выполнении операции произошло переполнение?
12. Что такое флаг, и для чего он нужен?
13. С помощью какой инструкции, и каким образом происходит сложение с учетом флага переноса?
14. С помощью какой инструкции, и каким образом происходит вычитание с учетом флага переноса?
15. Объясните назначение флагов переноса и нуля?
16. Объясните назначение флагов переполнения и знака?
17. Почему после загрузки DS и ES показывают не на начало сегмента данных? Куда они показывают?
18. Какие ошибки допущены при написании этой команду
    mov [ax+cx+4], [bx+c]
19. Как работает команда shl, rol

ЛАБОРАТОРНАЯ РАБОТА 9
ЦИКЛИЧЕСКИЕ И РАЗВЕТВЛЯЮЩИЕСЯ ПРОГРАММЫ

Цель работы: изучить команды и способы построения циклических и разветвляющихся программ.

Команда передачи, управления служит для передачи управления инструкции, не следующей непосредственно за данной. Управление может передаваться как внутри текущего сегмента кода (внутрисегментная передача управления), так и за его пределы (межсегментная передача управления). Тип передачи управления может быть задан ассемблеру предшествующим адресу перехода ключевым словом NEAR (внутрисегментная) или FAR (межсегментная).

Безусловные переходы. Инструкция безусловного перехода передаёт управление команде, адрес которой указан в инструкции. Команда безусловного перехода имеет вид

jmp [< тип > **ptr**] операнд.

<тип> - тип перехода **short** (короткий) – смещение 127 байтов вперёд или 128 байтов назад, **near** (близкий) – смещение в пределах сегмента (64 Кбайта), **far** (дальний) – в любой сегмент с любым смещением.

ptr – приставка, которую можно перевести как *указанный в*.

Если тип не задан, по умолчанию принимается **near**.

Всего можно выделить пять типов безусловных переходов (таблица 1).

Таблица 1

Типы команд безусловного перехода		
Название	Мнемоника	Описание
внутрисегментный прямой короткий	jmp short <операнд>	IP ← (IP) + 8-битное смещение, определяемое операндом
внутрисегментный прямой близкий переход	jmp near ptr <операнд>	IP ← (IP)+16-битное смещение, определяемое операндом
внутрисегментный косвенный переход	jmp <адрес операнда>	IP ← 16-битный адрес перехода

Межсегментный прямой далекий переход	jmp far ptr <операнд>	IP ← смещение операнда в сегменте CS ← адрес сегмента, содержащего операнд
Межсегментный косвенный далёкий переход	jmp far ptr <адрес операнда>	IP ← операнд CS ← адрес операнда +2

Условный переход. Команда условного перехода организует передачу управления при выполнении определённого в команде условия, в противном случае переход осуществляется на команду, следующую за инструкцией условного перехода. Условия определяются текущим состоянием флагов процессора. Каждая из 30 команд условных переходов проверяет определенную комбинацию флагов.

Все условные переходы являются короткими, т.е. адрес перехода должен отстоять не далее, чем на - 128 или +127 байтов от первого байта следующей команды.

Команды условной передачи управления и проверяемые при их выполнении условия приведены в таблице 2.

Таблица 2.

Инструкции условной передачи управления		
Мнемокод	условие перехода	
	Флаги	Смысл
ja/jnbe	CF or ZF=0	выше /не ниже и не равно
jae/jnb	CF=0	выше или равно/не ниже
jb/jnae	CF=1	ниже/не выше и не равно
jbe/jna	CF or ZF=1	ниже или равно/не выше
je/jz	ZF=1	равно/нуль
jne/jnz	ZF=0	не равно/не нуль
jg/jnle	(SF xor OF) or ZF=0	больше/не меньше и не равно
jge/jnl	SF xor OF=0	больше или равно/не меньше
jl/jnge	(SF xor OF)=1	меньше/не больше и не равно
jle/jng	((SF xor OF) or ZF)=1	меньше или равно/не больше
jp/jpe	PF=1	есть паритет/паритет четный
jnp/jpo	PF=0	нет паритета/паритет нечетный

jc	CF=1	перенос
jnc	CF=0	нет переноса
jo	OF=1	переполнение
jno	OF=0	нет переполнения
jns	SF=0	знак +
js	SF=1	знак -

Примечания:
1. термины "выше" и "ниже" применимы для сравнения беззнаковых величин (адресов);
2. термины "больше" и "меньше" используются при учете знака числа;
3. слова xor и or обозначают соответствующие логические операции.

Циклы. Инструкция, организующая программный цикл имеет вид:

loop[<условие повторения цикла>] <метка короткого перехода>

Инструкция **loop** использует содержимое регистра CX как счетчик повторений цикла. Команда **loop** уменьшает содержимое регистра CX на 1 и передает управление по адресу, определяемому меткой перехода, если содержимое CX ≠ 0, в противном случае выполняется следующая за LOOP инструкция. Подобно условным переходам инструкции этой группы могут осуществлять только короткие передачи управления, т.е. в пределах от -128 до +127.

Добавление к инструкции **loop** <условие повторения цикла> позволяет ввести дополнительные логические условия на повторение цикла:

loope/**loopz** – повторять, пока ноль;

loopne/**loopnz** – повторять, пока не ноль.

Проверка флага ZF осуществляется командой **loop**. Цикл повторяется, если содержимое CX ≠ 0 и выполняется соответствующее условие, в противном случае выполняется следующая за **loop** инструкция.

Пример выполнения работы

Дан масив из десяти слов, содержащих целые двухбайтовые числа со знаком mass dw 10,24,76,479,-347,281,-

24,70,124,97. Требуется найти максимальное значение в массиве.

Текст программы:

```
lea bx, mass          ; mass – адрес начала массива, например 0
mov cx, 10            ; Установить счетчик повторений цикла
mov ax, [bx]          ; Первый элемент массива в Аккумулятор
beg:   cmp [bx], ax   ; Сравнить текущий элемент
                      ; массива с максимальным
 jl no                ; он меньше
mov ax, [bx]          ; он больше или равен
no:    inc bx         ; Следующий элемент
inc bx                ; массива
loop beg
mov max, ax           ; max – адрес расположения найденного
                      ; максимального числа, например 0x40.
                      ; конец программы
```

Варианты заданий

Дан массив из десяти знаковых чисел (слов или байт). Требуется:

1. Найти количество отрицательных чисел. Массив байт.
2. Найти сумму всех положительных и отрицательных чисел. Массив слов.
3. Найти сумму абсолютных величин. Массив байт.
4. Найти количество положительных чисел. Массив байт.
5. Поменять местами пары соседних чисел. Массив слов.
6. Переставить числа в обратном порядке. Массив байт.
7. Заменить все отрицательные числа нулями. Массив байт.
8. Найти среднее арифметическое чисел. Массив слов.
9. Найти количество чисел больших 10h. Массив слов.
10. Найти наименьшее по абсолютной величине числа. Массив байт.
11. Найти наибольшее отрицательное число. Массив байт.
12. Найти произведение положительных элементов последовательности. Массив слов.

13. Найти среднее арифметическое квадратов ненулевых элементов последовательности. Массив слов.
14. Найти полусумму наибольшего и наименьшего чисел. Массив байт.
15. Найти среднее арифметическое отрицательных элементов последовательности. Массив слов.
16. Найти сколько в массиве чисел больше 12h и меньше 0Afh. Массив байт.
17. Найти есть ли в массиве два нуля, идущих подряд. Массив слов.
18. Найти сумму абсолютных величин, меньших 6. Массив байт.
19. Найти среднее арифметическое чисел больших 10. Массив слов.
20. Найти сколько чисел равно 12h. Массив байт.
21. Заменить все отрицательные числа их модулями. Массив байт.
22. Найти среднее арифметическое положительных чисел. Массив слов.
23. Найти количество чисел меньших 10h. Массив байт.
24. Найти наименьшее среди положительных чисел. Массив слов.
25. Найти наибольшее отрицательное число. Массив байт.

Контрольные вопросы

1. Для чего нужен префикс ptr ?
2. В чем отличие команд mov ax, offset mass и lea ax, mass?
3. В чем отличие команд mov ax, bx и mov ax, [bx]?
4. В чем отличие команд mov ax, [bp] и mov ax, [bx]?
5. В чем отличие команд mov ax, [bx+2] и mov ax [bx] + 2?
6. В чем отличие команд mov ax, [bx][si] и mov ax, [si][bx]?
7. Какие существуют разновидности инструкции jmp?
8. Как организовать межсегментную передачу управления?
9. Напишите фрагмент программы условного перехода к метке, лежащей от самого перехода на расстоянии 257 байт.
10. Для организации каких вычислений служат каманды loop, loope, loopne?

11. Модифицирует ли какие-нибудь регистры команда loop?
12. Можно ли организовать цикл по счетчику, не используя команды loop?
13. Можно ли организовать цикл while с помощью одной из команд loop?

ЛАБОРАТОРНАЯ РАБОТА 10
ПРИМЕНЕНИЕ ЛОГИЧЕСКИХ ИНСТРУКЦИЙ

Цель работы: изучить логические инструкции процессора.

Краткие теоретические сведения

Логические инструкции. Логические инструкции (команды) служат для сброса или установки отдельных бит в байте или слове. Они включают булевы операторы НЕ, И, ИЛИ, исключающее ИЛИ и операцию тестирования, которая устанавливает флаги, но не изменяет значения своих операндов.

not dst

Инструкция **not** инвертирует все биты байта или слова.

and dst, src

Инструкция **and** выполняет операции логическое И двух операндов (байтов или слов) и возвращает результат в операнд-приемник. Бит результата устанавливается в 1, если установлены в 1 оба соответствующих ему бита операндов, и устанавливаются в 0 противном случае.

or dst, src

Инструкция **or** выполняет операции логическое ИЛИ двух операторов (байтов или слов) и помещает результат на место операнда-приемника. Бит результата устанавливается в 1, если равен 1 хотя бы один из двух соответствующих ему битов операндов и устанавливается в 0 в противном случае.

xor dst, src

Инструкция **xor** выполняет операцию логическое исключающее ИЛИ двух операндов и помещает результат на место операнда-приемника. Бит результата устанавливается в 1, если соответствующие ему биты операндов имеют противоположные значения, и устанавливается в 0 в противном случае.

test dst, src

Инструкция **test** выполняет логическое И двух операндов (байтов или слов), модифицирует флаги, но результат не возвращает, т.е. операнды не изменяются.

В таблице 10.1. приведены значения регистра флагов, устанавливаемые логическими командами.

Таблица 10.1

Логические инструкции

Мнемокод		Флаги						*Действие*
Код	Операнды	O	S	Z	A	P	C	

Таблица 10.1

Логические инструкции

And	dst, src	0	x	x	u	x	0	логическое И
Or	dst, src	0	x	x	u	x	0	логическое ИЛИ
Xor	dst, src	0	x	x	u	x	0	логическое исключающее ИЛИ
Not	Dst	-	-	-	-	-	-	логическое НЕТ
Test	dst, src	0	x	x	u	x	0	логическое И без изменения dst

Примечание:

Флажок не модифицируется;

Устанавливается или сбрасывается в соответствии с результатом;

не определен;

Сбрасывается в 0.

Примеры использования логических команд.

1. Установить 3 и 0 биты в регистре **al**, остальные биты не изменять**.**

or **al**, 00001001b

2. Сбросить 4 и 6 битвы в регистре **al**, остальные биты не изменять.

and **al,** 10101111b

3. Инвертировать 2 и 4 биты в регистре al, остальные биты не изменять.

xor al, 00010100b

4. Перейти на метку LAB, если установлен 4 бит регистра al, в противном случае продолжить выполнение программы.

```
test al, 00010000b
jnz LAB
продолжаем
. . .
LAB:
```

5. Посчитать число единиц в регистре al, рассматривая байт, как набор бит.

```
mov cx, b              ; число сдвигов
xor bl, bl             ; обнуление BL
LL:     shl al, 1      ; сдвиг влево на один разряд
jnc NO                 ; переход, если нет переноса
inc bl                 ; иначе увеличить BL
NO:     loop LL        ; возврат, если cx ≠0
```

Пример выполнения работы

Дан массив из 10 байт, например,

NB db 04h, 07h, 14h, 23h, 04h,38h, 3Fh, 2Ah0Dh, 34h.

Все байты имеют нулевые старшие биты. Необходимо каждый байт содержащий единицу в нулевом бите дополнить до четного числа единиц установкой седьмого бита.

Текст программы:

```
lea bx, NB                  ; bx-текущий адрес массива NB
mov cx, 10                  ; cx-счетчик числа интераций
BEG:    mov al, [bx]        ; считать очередной байт массива
test al, 1b                 ; установлен ли бит 0?
jz BITOCLR                  ; нет, бит 0 сброшен
                            ; бит 0 установлен
  test al, 0ffh             ; четное число единиц?
jp OK                       ; да, больше ничего делать не надо
          or al, 80h        ; нечетное дополнить до четного?
jmp short OK                ; бит 0 сброшен
BITOCLR:  test al, 0ffh     ; четное число единиц?
jnp OK          ; нет, больше ничего делать не нужно
or al,80h       ; нечетное, дополнить до нечетного
OK:     mov [bx], al        ; записать измененный байт массива
loop BEG
QUIT:                       ; конец программы
```

Варианты заданий

1. Дан массив из 10 байт. Посчитать количество байт, в которых сброшены 6 и 4 биты.

2. Дан массив из 8 байт. Рассматривая его, как массив из 64 бит, посчитать количество единиц.

3. Дан массив из 8 байт. Рассматривая его как массив логических значений x0 x1 x2 x3 x4 x5 x6 x7 (true-есть ненулевые биты в байте, false-все биты нулевые), вычислить логическую формулу

f=(x7 & x6 & x1) V (x6 & x4 & x2 & x1 & x0) V (x7 & x6 & x3 & x1).

4. Дан массив из 10 байт. Посчитать количество байт с числом единиц в байте равным три.

5. Рассматривая байт как набор логических значений x7 x6 x5 x4 x3 x1 x0 (true-есть ненулевые биты в байте, false-все биты нулевые), вычислить логическую формулу

f=(x7 & x6 & x3) V (x6 & x4 & x2 & x1) V (x7 & x6 & x2 & x0)

6. Дан массив из 8 байт. Рассматривая его, как массив из 64 бит посчитать длину самой длинной последовательности единиц.

7. Дан массив из 10 байт. Посчитать количество единиц во всех разрядах, кратных трём: 3, 6, 9, …, 75, 78.

8. Дан массив из 5 байт. Рассматривая его как массив из 8 пятиразрядных слов, найти "исключающее или" всех 8 слов для выражения "10101".

9. Дан массив из 6 байт. Рассматривая его, как массив из 48 бит, посчитать в нём количество нулей.

10.Дан массив из 8 байт. Рассматривая его, как массив из 64 бит, посчитать количество пар единиц в окружении нулей. Конец последовательности рассматривать как нуль.

11. Дан массив из 7 байт. Рассматривая его, как массив из восьми семибитных слов, посчитать количество слов с нечетным числом нулей в слове.

12. Дан массив из 9 байт. Рассматривая его как массив из 72 бит, посчитать число переходов между нулями и единицами.

13. Дан массив из 3 байт. Рассматривая его, как массив из 24 бит, посчитать количество одиночных единиц в окружении нулей. Конец последовательности рассматривать как нуль.

14. Дан массив из 6 байт. Посчитать количество байт число единиц, в которых не превышает 3.

15. Дан массив из 11 байт. Посчитать количество байт, в которых нет единиц, стоящих рядом.

16. Дан массив из 4 байт. Рассматривая его, как массив из 32 бит посчитать длину самой длинной последовательности нулей.

17. Дан массив из 6 байт. Посчитать количество единиц во всех разрядах, кратных пяти: 5, 10, …, 45.

18. Дан массив из 3 байт. Рассматривая его как массив из 8 трёхразрядных слов, найти "исключающее или" всех 8 слов для выражения "101".

19. Дан массив из 7 байт. Рассматривая его, как массив из 56 бит, посчитать в нём количество нулей, стоящих после единицы. Конец последовательности рассматривать как нуль.

20. Дан массив из 8 байт. Рассматривая его, как массив из 64 бит, посчитать количество пар единиц в окружении нулей. Конец последовательности рассматривать как нуль.

21. Дан массив из 5 байт. Рассматривая его, как массив из восьми пятибитных слов, посчитать количество слов с чётным числом единиц в слове.

22. Дан массив из 6 байт. Рассматривая его, как массив из 48 бит, посчитать число двух единиц, стоящих между нулями. Конец и начало последовательности рассматривать как нули.

23. Дан массив из 3 байт. Рассматривая его, как массив из 24 бит, посчитать количество одиночных единиц в окружении нулей. Конец последовательности рассматривать как нуль.

24. Дан массив из 6 байт. Посчитать количество байт, число единиц в которых не превышает 3.

25. Дан массив из 11 байт. Посчитать количество байт, в которых нет единиц, стоящих рядом.

Контрольные вопросы

1. В чем отличие команд test и and?
2. Как сбросить 5-й бит переменной байта BB?
3. Как установить 5-й бит переменной байта BB?
4. Как инвертировать 5-й бит переменной байта BB?
5. Как проверить установлен ли 5-й бит переменной байта BB?
6. Как проверить четным или нечетным является количество установленных бит в байте?
7. Какие флаги условий модифицируются после выполнения команд and, or, xor?
8. В чем основное отличие команд логических и арифметических сдвигов?
9. Укажите максимальное число двоичных разрядов, на которые можно сдвинуть операнд с помощью одной команды сдвига?