

# Оглавление

<b>I</b>	<b>Что такое компьютер и зачем он нам нужен?</b>	<b>3</b>
<b>1</b>	<b>Как устроена современная электроника</b>	<b>4</b>
1.1	Двоичная система счисления . . . . .	4
1.1.1	Перевод числа из двоичной системы счисления . . . . .	5
1.1.2	Перевод числа в двоичную систему счисления . . . . .	5
1.2	Двоичная логика . . . . .	6
1.2.1	Задачи . . . . .	8
<b>2</b>	<b>Элементная база компьютеров</b>	<b>9</b>
2.1	Из чего состоит компьютер . . . . .	9
<b>3</b>	<b>Релейная логика</b>	<b>11</b>
3.1	Выключатели и бинарная логика . . . . .	11
3.2	Электромагнитное реле . . . . .	12
<b>II</b>	<b>Собираем релейный калькулятор</b>	<b>14</b>
<b>4</b>	<b>Релейный конструктор</b>	<b>15</b>
4.1	Модули . . . . .	15
4.2	Соединительные разъёмы . . . . .	15
<b>5</b>	<b>Первые эксперименты с конструктором</b>	<b>16</b>
5.1	Модуль переключателей . . . . .	16
5.1.1	Практикум . . . . .	16
5.1.2	Задачи . . . . .	17
<b>6</b>	<b>Хранение и передача данных</b>	<b>18</b>
6.1	RS-триггер . . . . .	18
6.2	Шина . . . . .	19
6.3	Шины в конструкторе . . . . .	20
6.4	Регистр . . . . .	20
6.4.1	Практикум . . . . .	21
6.4.2	Задачи . . . . .	22
6.5	Шина и регистровый файл . . . . .	22
6.5.1	Практикум . . . . .	22

<b>7</b>	<b>Вычисления</b>	<b>24</b>
7.1	Логические операции . . . . .	24
7.2	Отрицание . . . . .	24
7.3	Сдвиг . . . . .	24
7.4	Модуль унарных логических операций . . . . .	25
7.4.1	Практикум . . . . .	26
7.4.2	Задачи . . . . .	26
7.5	Логическое И . . . . .	26
7.6	Логическое ИЛИ . . . . .	27
7.7	Исключающее ИЛИ . . . . .	28
7.8	Модуль бинарных логических операций . . . . .	29
7.8.1	Практикум . . . . .	30
7.9	Арифметические операции . . . . .	33
7.10	Сложение . . . . .	33
7.11	Модуль сумматора . . . . .	35
7.11.1	Практикум . . . . .	36
7.11.2	Задачи . . . . .	37
7.12	Вычитание . . . . .	37
7.13	Вычитание через сложение с дополнительным обратным кодом . . . . .	37
7.14	Вычитание с помощью сумматора и инвертора . . . . .	38
<b>8</b>	<b>Калькулятор</b>	<b>40</b>
8.1	Чем калькулятор отличается от компьютера? . . . . .	40
8.2	Калькулятор для 7 операций . . . . .	40
8.3	Расширение вычислений до восьми бит . . . . .	41
<b>III</b>	<b>Строим релейный компьютер</b>	<b>43</b>
<b>9</b>	<b>Элементы компьютера</b>	<b>44</b>
9.1	Дешифратор . . . . .	44
9.1.1	Практикум . . . . .	44
9.2	ПЗУ . . . . .	44
9.3	Набор инструкций . . . . .	44
9.4	Декодер инструкций . . . . .	44
9.5	Тактовый генератор . . . . .	44
<b>10</b>	<b>Компьютер</b>	<b>45</b>

## Часть I

Что такое компьютер и зачем он  
нам нужен?

## Глава 1

# Как устроена современная электроника

Большинство современных электронных устройств представляют собой компьютеры. То есть они работают по какой-то программе. Обычно программа наблюдает за внешним миром и как-то реагирует на изменения в нём.

Например, «умная» лампочка загорается, когда становится темно. Лифт едет после нажатия кнопки. А мобильный телефон показывает смешную картинку, если человек нажал на правильное место на экране.

Все эти вещи кажутся очень разными, а на самом деле они очень похожи. Внутри каждого такого устройства есть маленький или большой компьютер. Компьютеры отличаются от других электрических и электронных устройств тем, что работают по программе, которую можно изменить, не переделывая всё устройство целиком.

Принципы работы всех компьютеров одинаковы: они работают с числами и программами. Программы состоят из последовательностей простейших операций над числами, которые быстро-быстро выполняются. Например, одним из чисел может быть температура воздуха (которую нужно измерить), а другим — напряжение на обогревателе, который нужно включить, если слишком холодно. Программисты записывают эти вычисления и процесс принятия решений в виде программы, а компьютеры затем программы выполняют.

Отличаются же компьютеры между собой тем, сколько чисел (то есть данных) они могут запоминать и с какой скоростью обрабатывать. Ну и конечно, такими незначительными деталями, как количество пикселей на экране или число кнопок у «мыши».

### 1.1 Двоичная система счисления

Так уж получилось, что компьютерам и другим цифровым устройствам удобнее всего работать в двоичной системе счисления. Люди привыкли работать с десятичной системой, а компьютеры работают в двоичной. И раз уж мы планируем позаниматься созданием цифровых игрушек, компьютеров и калькуляторов, стоит изучить двоичную систему счисления поподробнее.

Когда мы используем десятичную систему счисления, мы записываем числа с помощью десяти цифр, а у компьютеров числа «записаны» (если так вообще можно говорить) с помощью только двух цифр. Само число остаётся тем же самым независимо от формы записи. Двенадцать стульев будут двенадцатью стульями, если их записать как 12, XII

или  $1100_2$ .

Последний вариант это как раз двоичная запись (о чём нам намекает цифра 2 в конце). В ней используется только две цифры: 1 и 0. И этого вполне достаточно, чтобы закодировать любое число. 0 записывается как 0, 1 будет 1 (пока всё легко). Чтобы записать 2 разных цифр уже не хватает, поэтому нужно использовать больше разрядов. Значит 2 — это 10, 3 — 11, 4 — 100, 5 — 101.

Смысл тут такой же, как и в десятичном счёте. Там если разряд равен 9 (последняя доступная цифра), то при увеличении числа он заменяется на 0, а более старший разряд увеличивается на единицу. В двоичном счёте самая большая цифра это 1, поэтому перенос происходит, когда нужно такой разряд увеличить. Тогда при сложении 11 и 1 получается 100: сначала переполняется младший разряд, а затем из-за переноса и следующий за ним.

### 1.1.1 Перевод числа из двоичной системы счисления

Дальше мы будем часто встречать двоичные числа (например, наш калькулятор будет показывать именно двоичные числа). Но как их понимать? Как из двоичной записи получить десятичную? Можно просто запомнить все нужные значения. Например,  $1111_2 = 15_{10}$ . Но вдруг понадобятся числа больше чем из четырёх разрядов? Неплохо бы понять принцип.

Когда к числу добавляется ноль справа, это то же самое, что умножить исходное число на 10. Очевидно это так и для десятичной, и для двоичной систем счисления. Только в десятичной 10 это 10, а в двоичной 10 это 2.

Получается, что последовательным домножением единицы на  $10_2$  можно получать степени двойки:

$$1_2 = 2^0 = 1$$

$$10_2 = 2^1 = 2$$

$$100_2 = 2^2 = 4$$

$$1000_2 = 2^3 = 8$$

$$10000_2 = 2^4 = 16$$

А любое двоичное число можно разобрать на полученные выше слагаемые:

$$1101_2 = 1000_2 + 100_2 + 1_2 = 2^3 + 2^2 + 1 = 8 + 4 + 1 = 13$$

Один разряд двоичного числа называется битом. Если точнее, бит — это единица измерения количества информации. Он определяет одну из двух возможностей появления взаимно исключающих событий. А если эту информацию (выбранное событие) передать или записать где-то, то как раз можно использовать 1 или 0.

Два бита содержат информацию о четырёх событиях или состояниях (00, 01, 10, 11), три бита о восьми и так далее. Дальше мы будем в основном работать с четырёхбитными числами, то есть от 0000 до 1111 (или от 0 до 15).

### 1.1.2 Перевод числа в двоичную систему счисления

Если нам нужно в уме перевести десятичное число (например, 18) в двоичную систему счисления, то можно действовать так. Сначала найти наибольшую степень двойки, которая «помещается» в наше число (это будет 16). Потом вычитаем это число из исходного, а к результату добавляем единицу в нужном разряде (получится 2 и 10000). Повторяем эти два шага снова (результатом будет 0 и 10010). Исходное число обнулилось, ничего дальше делать не нужно.

Почему этот способ сработает? Не может ли дважды получиться единица в одном и том же месте? Оказывается нет, потому что степень двойки умноженная на два (если два раза вычитать одно и то же), это будет следующая степень двойки. А мы ведь вычитали наибольшую из возможных степеней, поэтому такого быть не может.

Этот способ неплох, но на практике часто делают наоборот — находят разряды результата начиная с младшего из них. Как найти самый младший разряд? Если посмотреть на двоичную запись, то становится очевидным, что этот разряд будет равен остатку от деления на  $10_2$ . Чтобы этот разряд теперь отбросить и перейти к следующему, всё число нужно разделить на  $10_2$ .

Делим  $18$  на  $10_2 = 2$ : младшая цифра (остаток) будет  $0$ , частное равно  $9$ . Теперь из  $9$  в результате деления получается цифра  $1$  и частное  $4$ . Найденные на текущий момент цифры двоичной записи это  $10$ .

Теперь на входе  $4$ . Остаток —  $0$ , частное  $2$ , двоичные разряды  $010$ . Из  $2$  получаем  $0, 1, 0010$ .

Остаётся единица, которая и становится старшим разрядом полученной двоичной записи:  $10010$ .

## 1.2 Двоичная логика

Компьютеры должны не только хранить числа, но и ещё что-то делать с ними. Люди обычно пользуются сложением, вычитанием, умножением и делением. Для процессора эти действия слишком сложные, поэтому они выполняются на основе более простых операций.

Работают элементарные операции с двоичными числами. То есть у каждой на входе одно или несколько чисел в двоичной форме, а на выходе обычно одно число (ну и может иногда ещё пара дополнительных битов).

Простейшие операции, из которых можно построить более сложные, вообще оперируют единичными битами. Например, два бита-операнда на входе и один бит-результат на выходе.

Чтобы описать, как работает операция, обычно приводят так называемую таблицу истинности. В ней для каждой комбинации значений входных операндов ( $A$  и  $B$ ) записан результат «вычислений» ( $R$ ). Примерно так:

A	B	R
0	0	0
1	0	1
0	1	1
1	1	1

Например, вторая строка таблицы означает, что если  $A = 1$  и  $B = 0$ , то результатом  $R$  будет  $1$ .

Вариантов таких операций (и соответственно таблиц) всего  $16$  (по числу комбинаций единиц и нулей на выходе). В реальности используются не все из них. Некоторые (самые полезные) мы сейчас рассмотрим.

Самая простая операция (или оператор) — это отрицание. У неё только один операнд и она меняет его значение на обратное. Записывать эту операцию мы будем как  $not\ A$ ,  $\neg A$  или  $\sim A$ .

A	$\sim A$
0	1
1	0

Логическое «или» (дизъюнкция, логическое сложение) выдаёт в качестве результата единицу, если хотя бы один из операндов равен единице (немного отличается от обычного сложения, потому что из 1 и 1 не получается 2). Обозначается такой оператор как  $A \text{ or } B$ ,  $A \vee B$  или  $A \mid B$ .

A	B	$A \mid B$
0	0	0
1	0	1
0	1	1
1	1	1

Логическое «и» (конъюнкция, логическое умножение) выдаёт единицу только если на обоих входах будет единица (и правда похоже на умножение). Обозначаться такой оператор может как  $A \text{ and } B$ ,  $A \wedge B$  или  $A \& B$ .

A	B	$A \& B$
0	0	0
1	0	0
0	1	0
1	1	1

Исключающее «или» (сложение по модулю 2, оператор неравенства) даёт нам единицу только для случаев, когда один операнд не совпадает с другим (то есть в одном единица, а в другом ноль). Второе название, сложение по модулю 2, означает, что результатом будет последняя цифра суммы операндов (в двоичном виде, естественно). Обозначается эта операция как  $A \text{ xor } B$ ,  $A \oplus B$  или  $A \wedge B$ .

A	B	$A \wedge B$
0	0	0
1	0	1
0	1	1
1	1	0

С помощью этих операций компьютеры могут вычислять принимать решения вроде «если число равно нулю» И «выполняется команда деления», то зажечь лампочку «ошибка». Конечно, это слишком элементарные операции (да ещё и с однокбитовыми числами). Как делать что-то сложнее, например сложение или вычитание, мы узнаем позднее.

Но нельзя не упомянуть ещё одну важную операцию, на которой строятся вычисления в современных компьютерах. Правда, в релейном конструкторе её нет, но эта технология не слишком-то современная.

Это операция «и-не», то есть логическое умножение в паре с отрицанием (ещё она называется штрихом Шеффера):

A	B	$\sim(A \& B)$
0	0	1
1	0	1
0	1	1
1	1	0

Оказывается, с помощью этой операции (и нескольких других, но сейчас о них не будем) можно выразить любую другую. Обозначим здесь «и-не» с помощью кружочка  $\circ$ :

- $\sim A = A \circ A$
- $A \& B = \sim(A \circ B)$
- $A \mid B = \sim A \circ \sim B$

### 1.2.1 Задачи

1. Придумайте как вычислять «исключающее или» используя только «и-не».



## Глава 2

# Элементная база компьютеров

### 2.1 Из чего состоит компьютер

Мы уже поняли, что внутри компьютеров спрятано много маленьких однобитных вычислителей. Но как компьютеру удаётся с помощью них делать что-то полезное?

Главное, что отличает компьютеры от других механизмов (или электронных устройств вроде часов) — это программы. Программа похожа на кулинарный рецепт: она состоит из последовательности простых шагов. В компьютере за работу по программе отвечает процессор. Именно он последовательно её «читает» и делает всё, что там написано. Фрагменты программы (шаги, инструкции, команды) обычно очень простые. Например, сложить два числа или вычислить «логическое и» (это мы уже умеем). Так как инструкций очень-очень много, а выполняются они быстро, современный компьютер успевает за незаметное человеческому глазу время посчитать координаты и цвета всех точек на экране, и приказать монитору их отобразить.

Кроме процессора в компьютере есть ещё память. Там хранятся все данные пользователя (музыка, фотографии, электронные письма), а также программы для обработки этих данных. Программа обычно хранится в памяти как последовательность команд. Чтобы двигаться по этим командам по мере их выполнения, нужно запоминать номер текущей. За это отвечает счётчик инструкций (program counter, instruction pointer). Счётчик инструкций обычно находится внутри процессора.

Процессор «смотрит» в память на текущую инструкцию и активирует один из модулей, отвечающий за её выполнение. Это может быть арифметическая операция, переход к другой части программы (тогда в счётчик инструкций попадает новое значение), передача данных на внешнее устройство (вывод пикселя на экран). После всего этого счётчик инструкций переходит к следующей команде.

Чтобы проделывать все эти шаги поочерёдно, используется тактовый генератор. Он с заданной частотой посылает сигналы на все остальные модули, в результате чего они активируются один за другим. Например, сначала процессор смотрит в память, потом декодирует инструкцию, то есть, разбирается с тем, какие именно действия нужно будет выполнять дальше, а после выполнения этих действий увеличивает счётчик инструкций.

Все числа (данные), с которыми работает процессор хранятся в памяти. Он может считывать их оттуда и записывать новые. Но внешняя (по отношению к процессору) память обычно работает не слишком быстро, поэтому в процессоре есть несколько регистров, используемых для вычислений. Например, процессор может сложить числа из двух регистров и записать результат в третий. А потом другая команда отправит значение этого регистра

во внешнюю память.

Мы будем строить собственный компьютер с помощью релейного конструктора. То есть формально это будет компьютер, но он будет настолько простой, что можно даже называть его процессором. А в качестве промежуточных этапов у нас будут получаться ещё более простые вычислители.

В целом, компьютер можно сделать из чего угодно: из шестерёнок, реле, транзисторов, микросхем. Электромагнитные реле хороши тем, что сразу понятно, что они делают. Современные компьютеры построены на транзисторах, которые упакованы в микросхемы, но с ними не так интересно играть. Например, реле можно переключить простой отвёрткой, а потом оценить, что изменилось.

Самое главное, что современные процессоры и компьютеры концептуально почти не отличаются от содержимого наших занятий. Мы будем создавать следующие части компьютера (и процессора):

1. Регистры для хранения данных.
2. Вычислительные модули для реализации команд.
3. Память программ.
4. Счётчик инструкций.
5. Логические схемы для декодирования команд и выбора правильных исполнительных модулей.
6. Тактовый генератор.

Такой компьютер сможет выполнять простые программы, например, найти остаток от деления одного числа на другое, используя только сложение и вычитание. А если в какой-то момент реле станет недостаточно, полученные знания можно будет применить для разработки более сложного процессора на современных технологиях или при программировании уже готовых устройств.

## Глава 3

# Релейная логика

### 3.1 Выключатели и бинарная логика

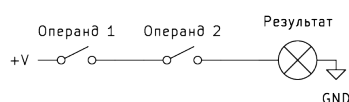
Первые компьютеры были построены на электромагнитных реле. А реле — это управляемый электрическим сигналом контакт. То есть это такие переключатели (вроде выключателей света, которыми мы пользуемся дома), но управляемые не нажатием вручную, а с помощью электрического сигнала.

Выключатели и электрический ток в проводах очень хорошо подходят для работы с двоичным кодом. Кнопка нажата — значит 1, а не нажата — 0. Ток течёт — 1, не течёт — 0.

Например, если на стене расположены 4 выключателя, то с их помощью можно закодировать четырёхбитное число. Переключив их в нужные положения мы «запишем» четырёхбитное число. А посмотрев на лампочки, соединённые с выключателями (условимся, что на выключатели смотреть нельзя), мы можем узнать, чтоб было записано.

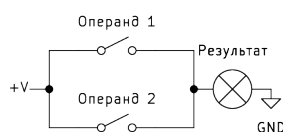
То есть запоминать и считывать данные мы можем. А как проводить вычисления? Хотя бы самые простые логические операции.

Например, мы хотим выполнять логическое «и»: если два выключателя нажаты, то лампочка загорается, а если хотя бы один разомкнут, лампочка гаснет. Значит операндами являются состояния выключателей (нажаты или нет), а результатом состояние лампочки (горит или нет). Этого можно достичь с помощью последовательного соединения:



Слева на схеме источник напряжения, а справа «земля», которая где-то за пределами рисунка соединена с минусовым контактом источника напряжения. Если ток будет течь слева направо через всю цепь, от плюса к минусу, он заставит светиться лампочку «Результат». Но ток потечёт, если только оба выключателя «Операнд 1» и «Операнд 2» будут замкнуты. Если незамкнутым выключателям сопоставить 0, а замкнутым 1, то мы получим, что эта схема выполняет функции логического «и».

Логическое «или» тоже сделать легко. Ведь ток должен течь в цепи если любой из выключателей-операндов нажат. Тогда соединив их параллельно мы сможем достичь такого результата:



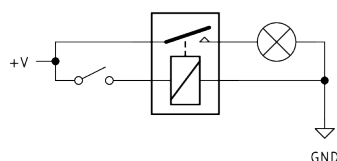
А что будет, если мы захотим объединить несколько операций в одной схеме? Например, выход со схемы «или» подать на вход схемы «и». Так как выход у нас это лампочка, а вход — выключатель, то придётся поставить туда человека, который будет смотреть на лампочку и нажимать на нужный выключатель в следующей схеме.

Вот если бы лампочка сама могла «нажимать» на выключатель... И это действительно возможно, только вместо лампочки нужно взять электромагнит. Так мы сделаем электромагнитное реле.

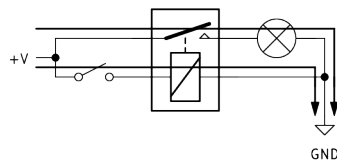
## 3.2 Электромагнитное реле

Электромагнитное реле — это электромагнит и один или несколько переключателей. При прохождении тока через обмотку электромагнита он притягивает якорь и нажимает на выключатели. С помощью этого механизма можно передавать сигналы с выхода одной схемы на вход другой.

На рисунке ниже схема, в которой есть одно реле (большой прямоугольник). Внутри этого реле находится электромагнит (перечёркнутый прямоугольник) и переключаемый контакт. К электромагниту подключена кнопка, позволяющий подавать на него напряжение, а к переключаемому контакту реле подключена лампочка.

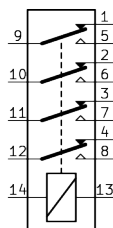


Когда кнопка не нажата, ничего не происходит. Ток не течёт ни через обмотку электромагнита, ни через лампочку. Если же кнопку нажать, на электромагните окажется напряжение и он притянет якорь и подвижную часть контакта, замкнув его. Таким образом через эта цепь тоже замкнётся и лампочка загорится.



При использовании электромагнитных реле для реализации логических операций в качестве логической единицы обычно используется наличие напряжения и протекание тока (цепь замкнута), а для логического нуля — отсутствие напряжения (цепь разомкнута).

В конструкторе применяются электромагнитные реле с четырьмя переключающими контактами. В электрических схемах мы используем нестандартное, но наглядное обозначение таких реле:

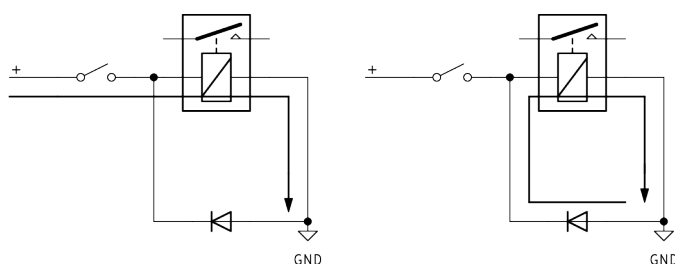


Контакты 13 и 14 подключены к катушке, управляющей состоянием контактов. Когда напряжение на катушку не подано (и ток через неё не течёт), контакты переключателей соединены так: 9 – 1, 10 – 2, 11 – 3, 12 – 4.

Если же реле включится, подвижная часть контактов сдвинется, разомкнув цепи, описанные выше, и замкнув следующие: 9 – 5, 10 – 6, 11 – 7, 12 – 8.

В каждое реле конструктора встроен светодиод, поэтому можно легко увидеть, когда оно включается. Это помогает отлаживать схемы или наблюдать хранящиеся в релейной памяти значения.

Во всех модулях параллельно каждой из катушек реле подключён диод в обратном направлении. Это сделано для предотвращения искрения контактов других реле при их размыкании. Искрение возникает из-за того, что несмотря на разрыв цепи из-за индуктивности катушки ток должен продолжать течь какое-то время. Но из-за разрыва это невозможно в обычном режиме, поэтому возникает электрическая дуга, постепенно разрушающая контакты.



Чтобы контакты меньше портились и нужен диод. Оставшийся электрический импульс проходит уже через него, а не через разомкнутые контакты.

## Часть II

# Собираем релейный калькулятор

## Глава 4

# Релейный конструктор

### 4.1 Модули

Релейный конструктор включает не просто реле и провода, как можно было бы подумать, а несколько видов готовых модулей, выполняющих простые функции. Каждый модуль состоит из печатной платы, нескольких разъёмов на ней и других компонентов, в зависимости от его функций (реле, переключатели, диоды, резисторы).

Это похоже на простые микросхемы: один модуль может складывать числа, другой хранить несколько битов информации, и так далее. Ещё есть модуль с переключателями, чтобы человек мог управлять схемой.

В следующих главах мы будем постепенно знакомиться с каждым из модулей, соединять их друг с другом, чтобы делать вычислительные устройства: несложные логические схемы, игрушки, калькулятор и даже компьютер.

### 4.2 Соединительные разъёмы

Модули соединяются между собой с помощью разъёмов по краям плат или плоских кабелей. Часто они расположены парами, чтобы можно было либо стыковать платы, либо пользоваться кабелем. В каждом разъёме для подключения платы или кабеля есть контакты для питающего напряжения (контакт 1) и нуля (контакт 8). Поэтому все разъёмы электрически совместимы друг с другом, но логически их соединение не всегда имеет смысл.

Кроме питающего напряжения через соединения передаются и полезные сигналы, занимающие до четырёх контактов (номера от 2 до 5), в зависимости от назначения модуля. Они используются для кодирования четырёхбитного числа или передачи четырёх управляющих сигналов разного назначения.

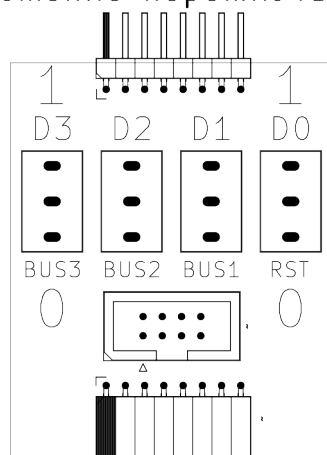
Контакты 6 и 7 в текущей версии конструктора не используются.

## Глава 5

# Первые эксперименты с конструктором

### 5.1 Модуль переключателей

Положение переключателей



Положение переключателей

Модуль с переключателями (или тумблерами) используется для установки уровней 0 или 1 на выходном разъёме вручную. Присоединяя его к разным разъёмам, можно задавать четырёхбитное число, либо переключать до четырёх управляющих сигналов.

Проще всего проверить работу тумблеров, подключив их к управляющей шине регистрового модуля, в который вставлены только 4 реле.

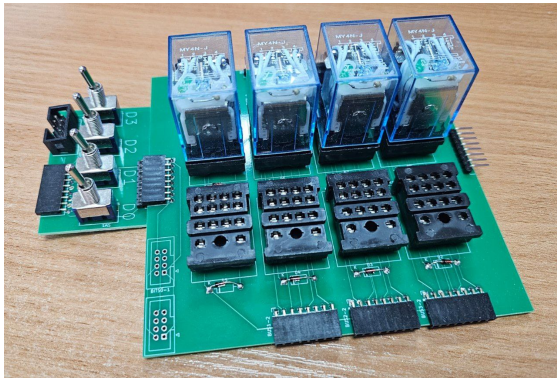
Входных разъёмов у этого модуля нет, только выходные, на которые попадают сигналы с тумблеров. Все разъёмы подключены параллельно, поэтому можно синхронно управлять одним набором переключателей сразу несколькими модулями, если подключить их все.

#### 5.1.1 Практикум

Список модулей:



- Модуль переключателей: 1 штука
- Регистровый модуль как шинный формирователь: 1 штука



1. Переключать тумблеры. Убедиться, что положение одного тумблера меняет состояние одного реле.
2. Запомнить включенное и выключенное состояния тумблеров, а также их порядок по отношению к реле, чтобы позднее не было проблем с управлением другими схемами.

### 5.1.2 Задачи

1. Придумать, как соединить тумблеры для выполнения логического ИЛИ.
2. Придумать, как соединить тумблеры для выполнения логического И.

## Глава 6

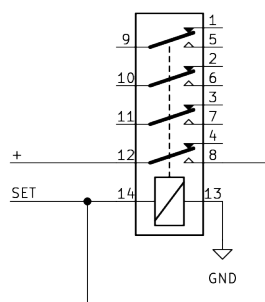
# Хранение и передача данных

### 6.1 RS-триггер

Триггер — это простейшее устройство для хранения одного бита информации. Если он включен, там хранится единица, а если выключен — хранится ноль.

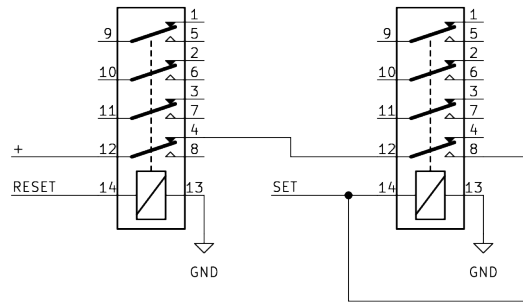
Хранение одного бита можно сделать с помощью одного реле. Если оно включается с помощью внешнего сигнала, то реле «сохраняет» единицу. Но когда внешний сигнал пропадает, сохранённая единица пропадает тоже, потому что на катушке реле больше нет напряжения.

Чтобы включённое состояние запоминалось, нужно организовать обратную связь через один из контактов реле. Как только реле включается с помощью входа *SET*, на его обмотку подаётся напряжение через его же замкнутый контакт. Поэтому даже если перестать подавать напряжение на вход *SET*, реле всё равно останется включённым:



Но если такое реле уже включилось, выключить его не получится. Единственный способ — это отключить питание всей схемы. Тогда контакты разомкнутся и при повторной подаче питания реле будет в выключенном состоянии.

Чтобы не отключать всю схему, нужно предусмотреть отдельное реле, позволяющее отключить питание только для запоминающего реле:



Теперь подав сигнал на вход *RESET*, можно сбросить наше реле-триггер. И там опять окажется ноль.

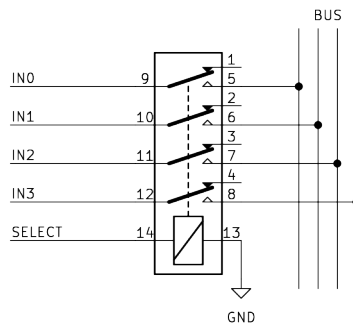
Триггер с двумя входами *RESET* и *SET* называется RS-триггером. На практике удобно объединить несколько таких триггеров в многобитовый регистр и использовать для них единый сигнал сброса, сэкономив несколько реле.

## 6.2 Шина

Шина — это набор проводников, объединённых общим назначением. К этим проводникам подключаются несколько устройств для обмена сигналами.

Например, по шине данных процессор может читать или записывать данные из памяти или жёсткого диска. Шина адреса используется для выбора определённой ячейки в памяти. А по шине управления приходят сигналы, позволяющие отличать обращение к памяти от обращения к диску.

Так как к шине обычно подключается больше одного входа и больше одного выхода, необходимо изолировать лишние устройства, когда шина используется другими. В релейном компьютере мы можем использовать для подключения к четырёхбитной шине шинный формирователь из одного реле:



Когда такое реле включается, оно может соединить какое-то устройство (например, регистр) с шиной. Подключение будет двунаправленным, поэтому с помощью этой схемы можно как записывать данные в регистр, так и считывать оттуда.

Шинные формирователи подключаются к каждому устройству, поэтому для передачи данных и вычислений требуется формировать множество сигналов *SELECT*.

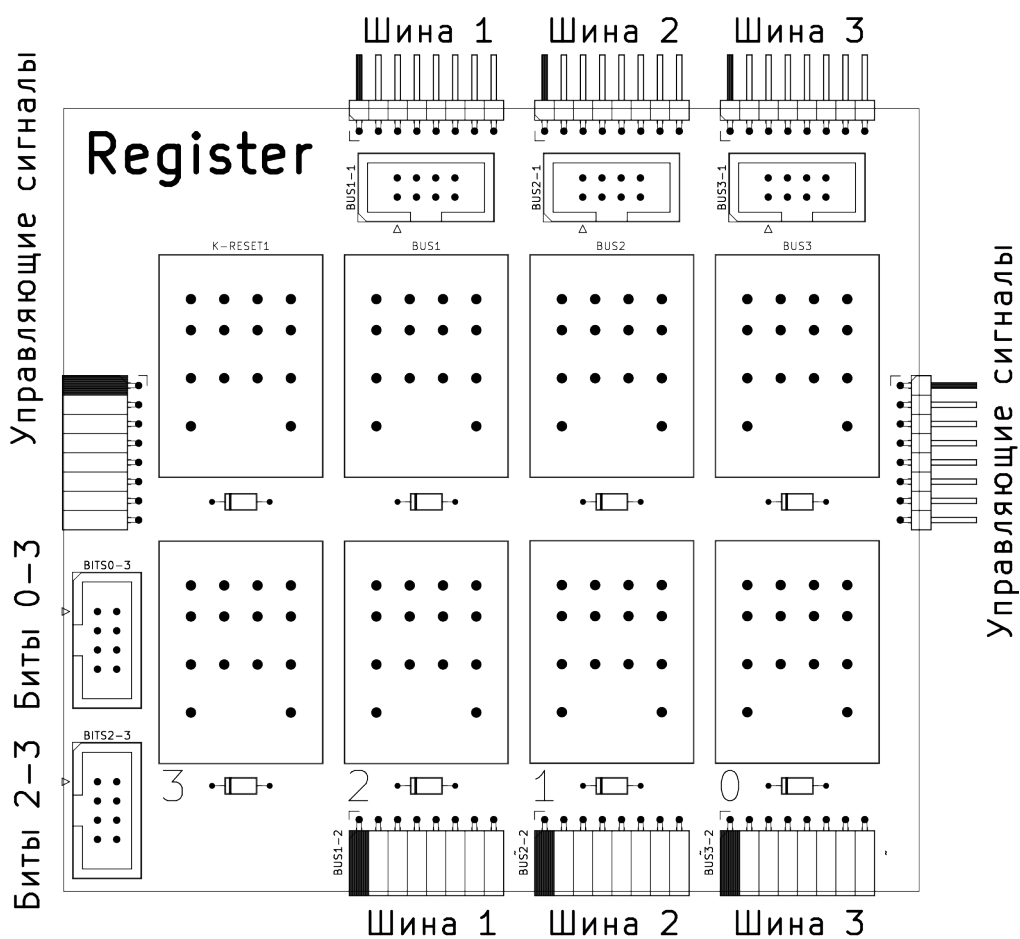
### 6.3 Шины в конструкторе

В конструкторе все модули могут подключаться к шинам по которым передаются от 1 до 4 сигналов.

Например, 4 сигнала могут использоваться для управления модулем регистра. Или для передачи данных из одного регистра в другой. Каждым из таких сигналов можно управлять с помощью модуля переключателей, соединив его с соответствующей шиной.

### 6.4 Регистр

Регистры внутри процессоров (или периферийных устройств) используются для хранения данных и для проведения вычислений. Регистры могут иметь выделенное назначение (например, хранить режим работы устройства) или же используются почти для всего (адрес, операнд арифметических операций, значение для ввода-вывода).



Модуль четырёхбитного регистра состоит из четырёх реле-триггеров, одного реле для обнуления регистра и трёх реле для подключения к шинам данных.

Для хранения данных используются RS-триггеры, описанные выше, но сигнал для сброса у них общий. Таким образом, записывать данные можно только во все четыре бита сразу.

Поэтому такой модуль и реализует функции цельного регистра, а не нескольких разрозненных RS-триггеров.

Также модуль регистра содержит три шинных формирователя. Их можно использовать для подключения триггеров к разным устройствам через три шины. Например, одна шина может быть предназначена для первого операнда при вычислениях, вторая для второго операнда, а третья для копирования данных между регистрами.

Если на плату регистра установить только реле шинных формирователей, получится модуль, который может подключать четырёхбитные сигналы (полученные из разъёма для прямого чтения значений триггеров) к одной из выбранных шин.

Модуль регистра имеет следующие разъёмы:

- Слева и справа: управляющие сигналы сброса и выборки. Можно подключить тумблеры для ручного включения сигналов. Также можно соединить несколько модулей регистра, чтобы управлять одним набором сигналов сразу для 8, 12 ... бит.
- Сверху и снизу: три шины данных. Реле регистра могут подключаться к шинам для записи или чтения данных.
- Дополнительные разъёмы с битами 0 – 3 и 2 – 3 для чтения или записи значения без подключения к шине.

Чтобы прочитать значение регистра нужно активировать сигнал выборки его на нужную шину. А уже через шину сигналы поступят на вход какой-то другой схемы.

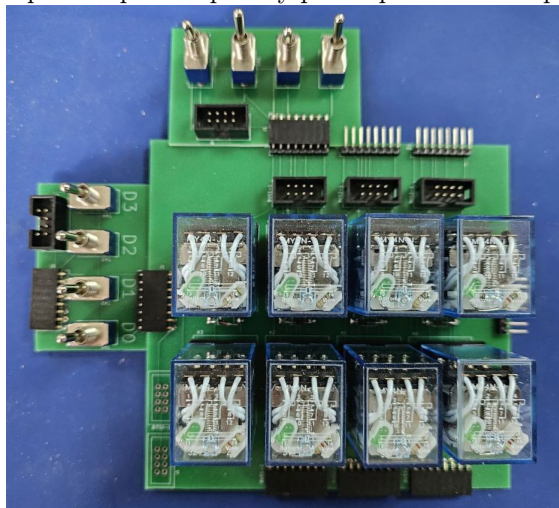
Для записи значения в регистр нужно сначала обнулить его триггеры, иначе запись не произойдёт в тех битах, где уже хранятся единицы. После этого регистр подключается к нужной шине, и триггеры защёлкивают нужное значение.

### 6.4.1 Практикум

Список модулей:

- Модуль переключателей: 2 штуки
- Регистровый модуль: 1 штука

Протестировать работу регистра можно собрав следующую схему:



- Тумблеры слева управляют работой регистра. Бит 0 — обнуление, бит 1 — выборка на шину 1.
- Тумблеры сверху нужны для ввода значения регистра. Когда он подключается к шине 1, значения, набранное на тумблерах, записывается в регистр.

#### Регистр без шины

1. Подключить тумблеры проводом к битам 0 — 3 вместо шины.
2. Набирать значение, убедиться, что биты переключаются в 1, но не возвращаются в 0.
3. Обнулить тумблеры с данными.
4. Включить и выключить сигнал сброса. Убедиться, что значения всех битов теперь 0.

#### Регистр с шиной

1. Отключить все управляющие сигналы.
2. Набрать значение на тумблерах для данных. Убедиться, что это не влияет на регистр.
3. Включить и выключить сигнал выборки на шину 1. Убедиться, что данные записались в регистр.
4. Включить и выключить сигнал сброса. Убедиться, что значения всех битов теперь 0.

#### 6.4.2 Задачи

1. Придумайте, как скопировать данные из одного регистра в другой, не запоминая их в голове.

### 6.5 Шина и регистровый файл

Несколько регистров можно соединить в регистровый файл. Регистровый файл — это набор регистров. Он может быть однородным (все регистры эквивалентны) или нет (разные регистры имеют разное назначение).

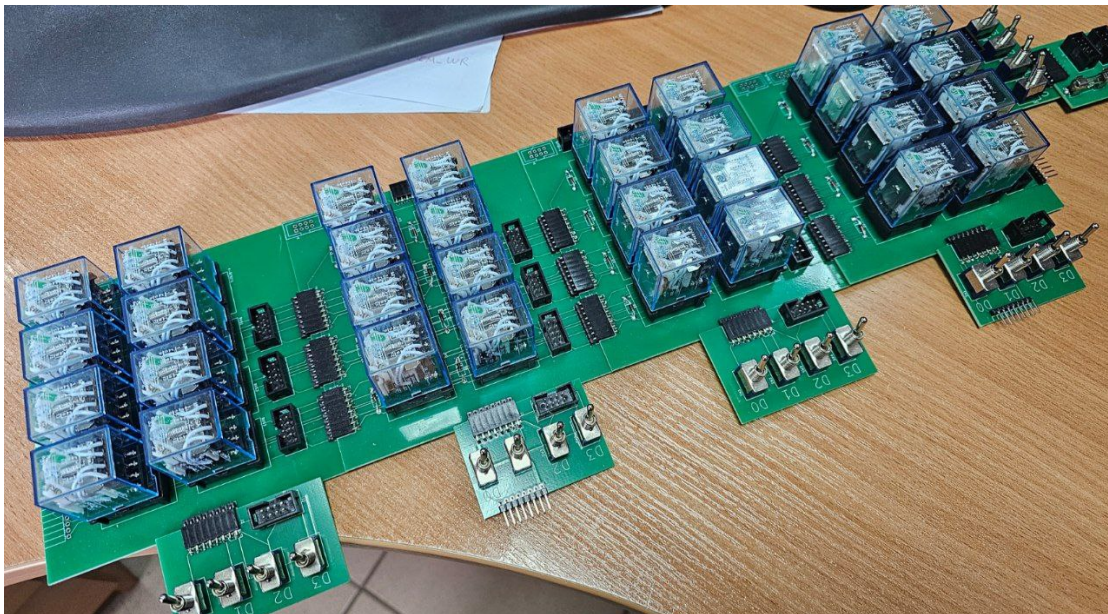
У каждого из регистров есть сигналы выборки на одну из трёх шин. Если два регистра подключены к одной шине одновременно, то значения одного будут копироваться в другой. Если точнее, включённые биты включают аналогичные в другом регистре, то есть копирование возможно в обе стороны одновременно.

Нулевые биты при этом копироваться не могут. Для записи нулей регистр необходимо сбросить.

#### 6.5.1 Практикум

Список модулей:

- Модуль переключателей: 5 штук
- Регистровый модуль: 4 штуки



Запись в регистры:

1. Отключить все управляющие сигналы.
2. Набрать значение на тумблерах, подключённых к шине данных 1.
3. Подключить с помощью тумблера регистр к шине 1. Убедиться, что в него записалось набранное значение.
4. Отключить регистр от шины.
5. Подключить другой регистр к шине 1. Убедиться, что в него записалось набранное значение.

Копирование значения:

1. Отключить все управляющие сигналы.
2. Подключить регистр с ненулевыми битами к шине 2.
3. Подключить пустой регистр к шине 2. убедиться, что он получил такое же значение, что и в первом регистре.
4. Отключить все управляющие сигналы.
5. Аналогично проверить шину 3.

## Глава 7

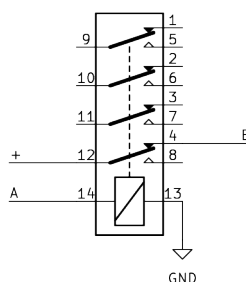
# Вычисления

### 7.1 Логические операции

### 7.2 Отрицание

Одна из самых простых операций, которая используется в современных компьютерах — это операция отрицания. Она преобразует единицу в ноль, а ноль в единицу.

Такую логику можно реализовать с помощью нормально замкнутого контакта реле:



Когда реле выключено, ток может течь через контакт, поэтому на выходе  $B$  получается единица. Если же на входе  $A$  появляется единица, реле переключается и ток через нормально замкнутый контакт не течёт. На выходе  $B$  теперь ноль.

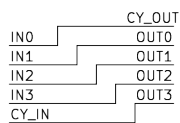
### 7.3 Сдвиг

Операция сдвига ещё более простая по сравнению с отрицанием. Суть её в том, что биты в числе перемещаются по какому-либо принципу (влево, вправо, по кругу).

В конструкторе реализован сдвиг вправо. Нулевой бит вытесняется из числа (и может использоваться отдельно). Первый бит становится нулевым, второй первым и так далее. Такой сдвиг эквивалентен делению на два, поэтому деление на степень двойки в программах процессор вычисляет с помощью инструкции сдвига.

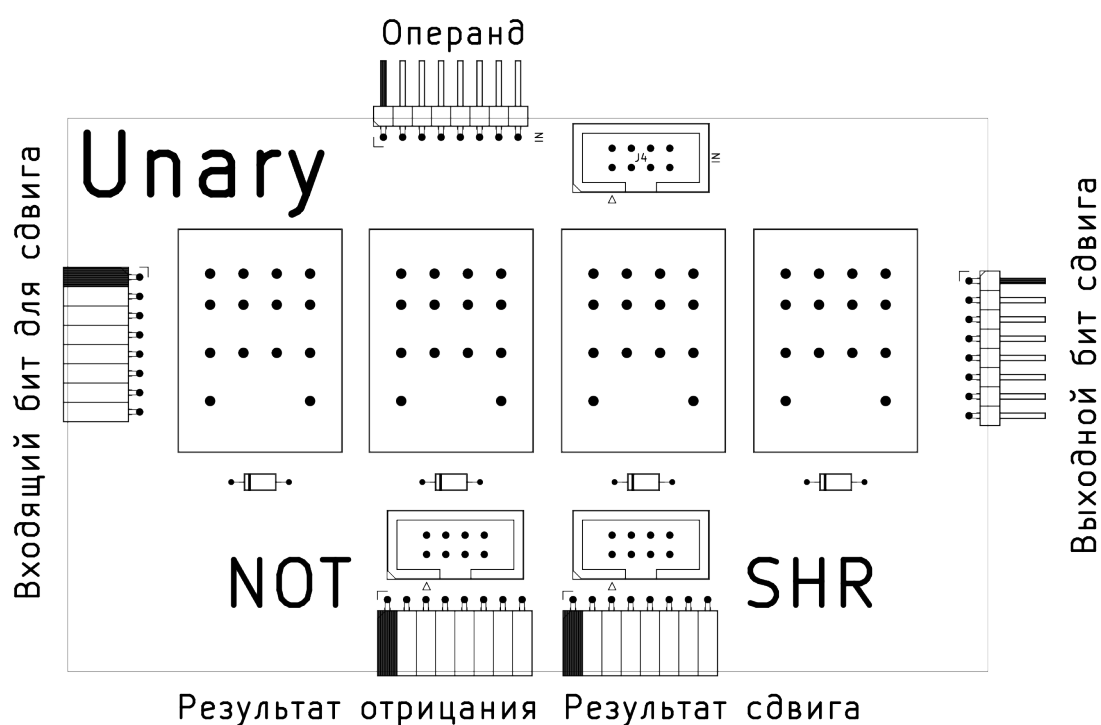
Для реализации сдвига вправо можно даже не использовать реле:





В конструкторе есть готовый модуль только для сдвига вправо. Ещё в вычислениях часто используется сдвиг влево, но его легко реализовать с помощью сложения числа с самим собой.

## 7.4 Модуль унарных логических операций



Модуль для унарных операций выполняет действия над 4-битным числом: сдвиг вправо и инверсия битов. Он имеет следующие разъёмы:

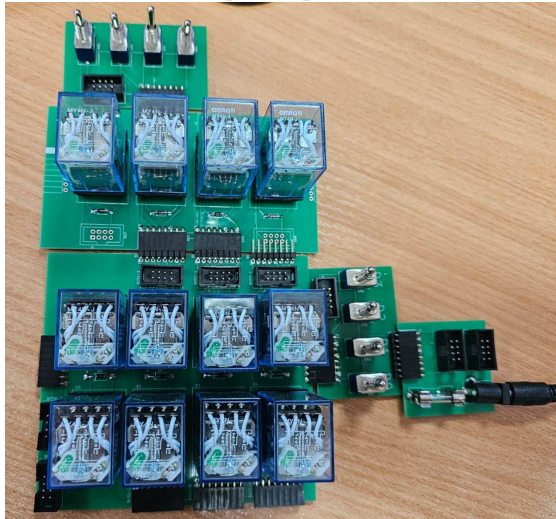
- Слева и справа: шины для каскадирования нескольких модулей. Можно использовать, когда нужен сдвиг чисел больше четырёх бит.
- Сверху: разъёмы для подключения сдвигаемого операнда. Можно подсоединить к какой-нибудь шине, а можно к модулю переключателей или регистру.
- Снизу: два разъёма с выходными сигналами. На один поступает инвертированный операнд, а на другой сдвинутое вправо значение операнда.

### 7.4.1 Практикум

Список модулей:

- Модуль переключателей: 2 штуки
- Модуль унарных операций: 1 штука
- Регистровый модуль: 1 штука

На вход модуля унарных операций подключается модуль с тумблерами. Выходы подключаются к шинам регистра.



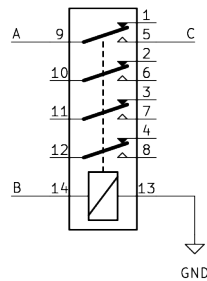
1. Отключить все управляющие сигналы.
2. Набрать на тумблерах со входными данными значение 1100.
3. Подключить выходной регистр к шине 1. Убедиться, что в него записалось значение 0011 (инверсия).
4. Отключить регистр от шины, сбросить его значение.
5. Подключить выходной регистр к шине 2. Убедиться, что в него записалось значение 0110 (сдвиг вправо).

### 7.4.2 Задачи

1. Собрать устройство, позволяющее сдвигать содержимое регистра и записывать результат обратно. Занести в регистр значение 1000 и сдвигать его до тех пор, пока не получится 0001. Вторую часть можно выполнять на скорость.

## 7.5 Логическое И

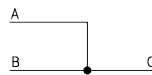
Реле представляет собой управляемый выключатель или переключатель. Если на один контакт  $A$  выключателя подать сигнал, то на другом  $C$  он появится только если на обмотке реле  $B$  есть напряжение (логическая единица).



Получается, что на выходе выключателя сигнал будет только тогда, когда реле включено и на входе выключателя тоже не ноль. Это означает, что такая схема реализует операцию логическое «И»:  $C = A \wedge B$ .

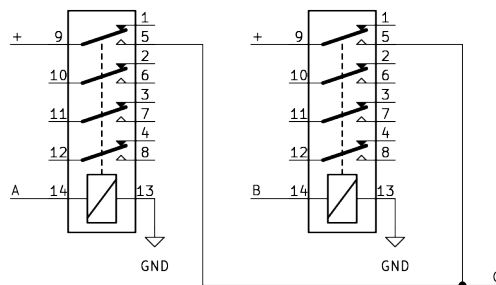
## 7.6 Логическое ИЛИ

Схему для выполнения логического «ИЛИ» можно получить, если соединить два выхода (две цепи). Тогда напряжение на объединённом выходе появится в любом из вариантов, если на первом выходе единица или на втором:  $C = A \vee B$ .



Но у этого способа есть один недостаток. Если на входе  $A$  окажется единица, то так как все входы и выходы соединены в одну цепь, поданное через  $A$  напряжение будет влиять и на вход  $B$ . И когда  $B$  используется ещё где-то, всё заработает неправильно, какое-то реле включится. Такой вот паразитный сигнал —  $A$  влияет на выходы, зависящие от  $B$ .

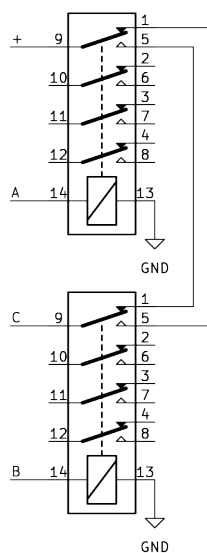
Поэтому иногда стоит использовать схему с реле для логического «ИЛИ»:



## 7.7 Иключающее ИЛИ

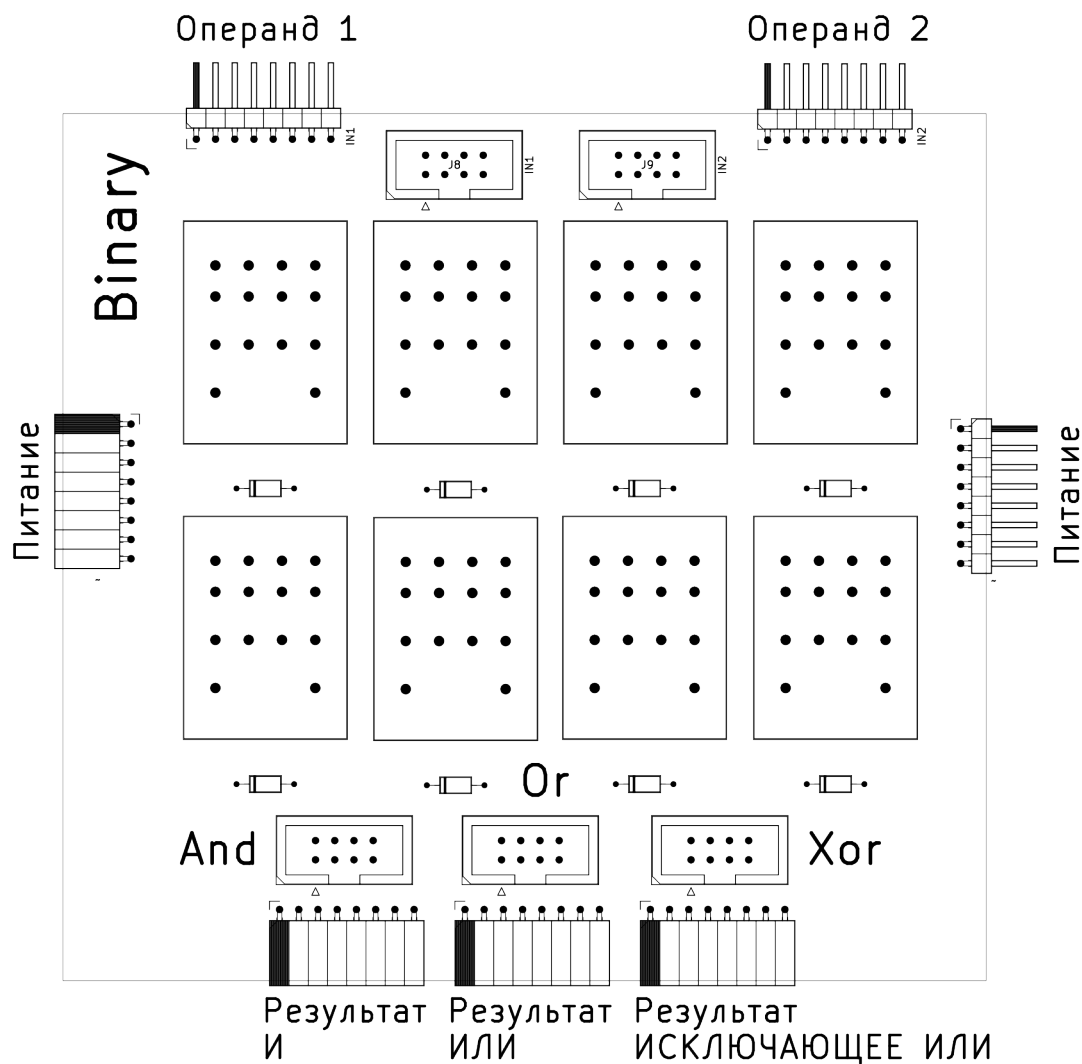
Схема для вычисления «исключающего или» несколько сложнее. Результат операции должен равняться единице только тогда, когда операнды не равны. То есть если на одном из входов уже было напряжение, а потом оно появляется и на другом входе, выход из состояния единицы должен переключиться в ноль.

Такую логику проще всего реализовать с помощью двух переключающих контактов:



Благодаря перекрёстному соединению переключателей проводниками, сигнал на выходе *C* появляется только тогда, когда переключатели на реле находятся в противоположных положениях.

## 7.8 Модуль бинарных логических операций



Модуль логических операций поддерживает вычисление AND, OR и XOR. У модуля есть два четырёхбитных входа и три выхода. Каждый выход отвечает за одну операцию.

Модуль имеет следующие разъёмы:

- Слева и справа: шины для каскадирования нескольких модулей. Полезных сигналов (как при сдвиге) через них не передаётся, но можно их использовать для фиксации нескольких логических модулей вместе, когда нужны более вычисления разрядностью больше четырёх бит.
- Сверху: разъёмы для подключения сигналов-операндов. Можно подсоединить к шинам, а можно к модулям переключателей или регистрам.
- Снизу: три разъёма с выходными сигналами: результатами вычисления операций И, ИЛИ, Исключающее ИЛИ.

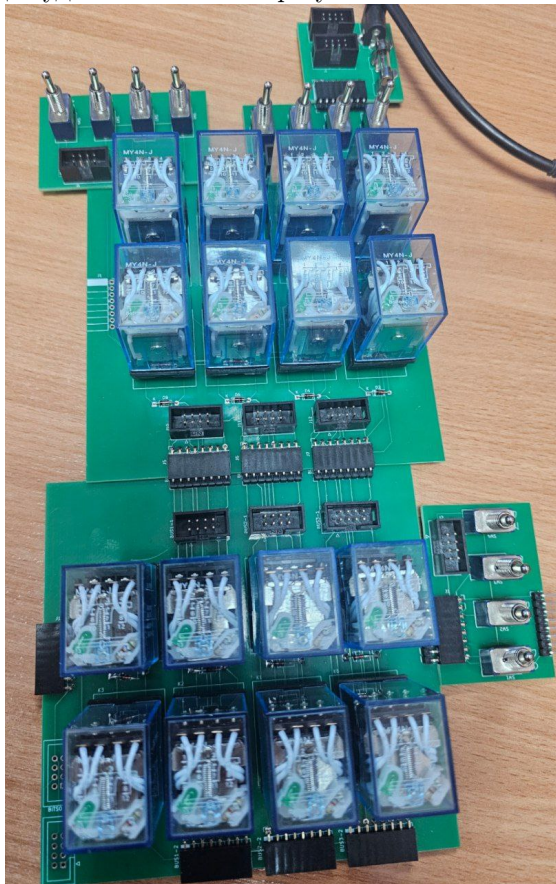
### 7.8.1 Практикум

#### Работа модуля

Список модулей:

- Модуль переключателей: 3 штуки
- Модуль логических операций: 1 штука
- Регистровый модуль: 1 штука

Ко входам модуля подключаются два модуля с тумблерами, а к выходам — регистр, куда будет записываться результат.



1. Отключить все управляющие сигналы.
2. Набрать на тумблерах первого операнда значение 1100.
3. Набрать на тумблерах второго операнда значение 1010.
4. Подключить выходной регистр к шине 1. Убедиться, что в него записалось значение 1000 (AND).
5. Отключить регистр от шины, сбросить его значение.

6. Подключить выходной регистр к шине 2. Убедиться, что в него записалось значение 1110 (OR).
7. Отключить регистр от шины, сбросить его значение.
8. Подключить выходной регистр к шине 3. Убедиться, что в него записалось значение 0110 (XOR).

### Управление лампочкой с помощью нескольких выключателей

Управлять одной лампочкой может быть очень полезным, если эта лампочка освещает длинный коридор. Эту задачу можно решить с помощью механических переключателей, но мы попробуем собрать аналогичную логическую схему.

В качестве лампочки будем использовать одно из управляющих реле регистра модуля. Схема должна позволять в любой момент зажечь или выключить лампочку переключением любого из трёх тумблеров.

Построим таблицу истинности для схемы из выключателей и лампочки: входами  $A$ ,  $B$ ,  $C$  будут положения тумблеров, а выходом  $R$  — состояние лампочки.

Сначала всё выключено, поэтому лампочка тоже не горит:

A	B	C	R
0	0	0	0

Если один из тумблеров переключить, лампочка загорится:

A	B	C	R
0	0	0	0
1	0	0	1
0	1	0	1
0	0	1	1

Потом можно выключить уже нажатый тумблер (и вернуться к первой строке таблицы), а можно «дойти до конца коридора» и переключить другой. Лампочка должна погаснуть:

A	B	C	R
0	0	0	0
1	0	0	1
0	1	0	1
0	0	1	1
1	1	0	0
1	0	1	0
0	1	1	0

И остаётся вариант, когда переключены все три тумблера. Так как число переключений нечётное, лампочка загорается снова:

A	B	C	R
0	0	0	0
1	0	0	1
0	1	0	1
0	0	1	1
1	1	0	0
1	0	1	0
0	1	1	0
1	1	1	1

Такая таблица истинности соответствует операции «исключающее или», только для трёх операндов вместо двух. Можете убедиться самостоятельно, что таблица описывает такое выражение:

$$R = A \text{ xor } B \text{ xor } C = (A \text{ xor } B) \text{ xor } C = A \text{ xor } (B \text{ xor } C)$$

Чтобы его «посчитать» с помощью модулей конструктора нужно немного подумать. Логический модуль считает *xor* для пары четырёхбитных операндов. А у нас тут есть три однобитных. То есть нужно либо взять два логических модуля, либо подавать сигнал с выхода одного однобитного *xor* на вход другого внутри одного и того же модуля.

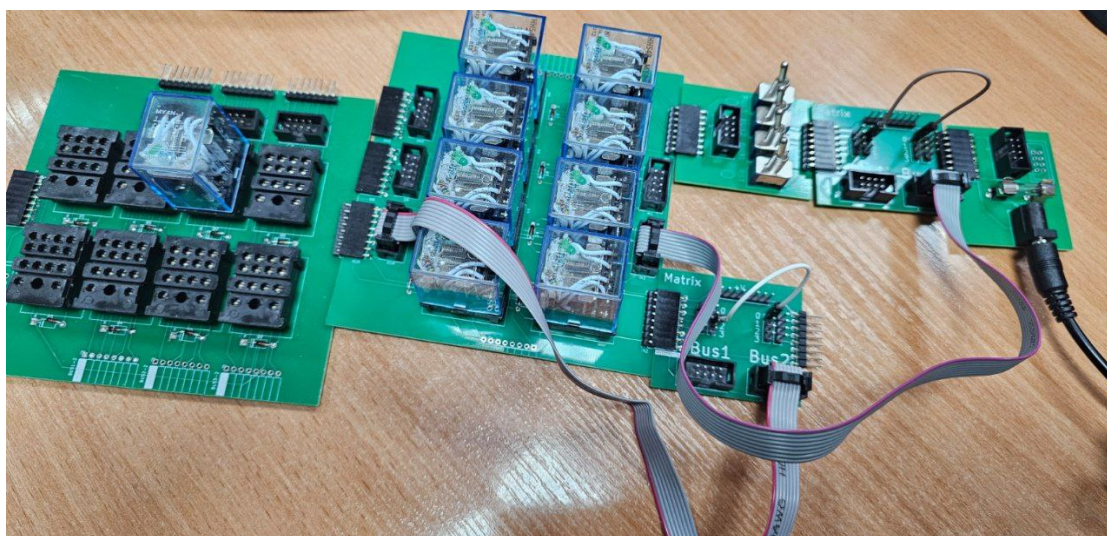
Первый вариант можно реализовать с помощью такой схемы:



Тут реализуется формула, приведённая выше: сначала выполняется  $A \text{ xor } B$ , а потом с полученным результатом делается  $\text{ xor } C$ . Каждый из входных битов расположен на своей плате переключателей. Значение выходного бита показывается с помощью реле на плате регистрового модуля.

Чтобы использовать только один логический модуль, понадобится пара кросс-модулей для перемещения битов с одного места на другое:





Здесь входные данные задаются с помощью трёх тумблеров: 0 –  $A$ , 1 –  $B$ , 2 –  $C$ . Все сигналы с них подаются на первый вход логического модуля. Но сигнал  $B$  мы тут не используем, а с помощью коммутационной матрицы перенаправляем его на нулевой бит второго входа логического модуля.

Получается, что нулевой бит на выходе будет равен  $A \text{ xor } B$ . С помощью второй матрицы мы передаём этот сигнал на второй бит второго входа. В результате второй бит будет равен  $(A \text{ xor } B) \text{ xor } C$ , а чтобы это увидеть, к выходу  $\text{xor}$  также подключен регистровый модуль с одним управляющим реле.

### Игра «Кто быстрее»

### Игра «Волк, коза и капуста»

Тумблеры показывают положение волка, козы, капусты и человека. Схема показывает ошибку, если позиция запрещённая.

### Шифратор и дешифратор на логических элементах

### Умножение двух двухбитных чисел

## 7.9 Арифметические операции

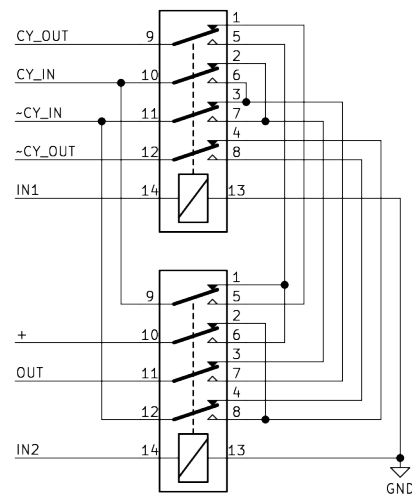
### 7.10 Сложение

Чтобы складывать числа, сначала нужно научиться складывать отдельные биты. Сумма двух битов может дать результат  $0_2$ ,  $1_2$  или  $10_2$ . То есть при сложении получается уже двухбитовое число.

Схему сложения удобнее всего строить из однотипных компонентов, складывающих фиксированное количество битов. На выходе такого компонента будет результат сложения, а также бит переполнения (переноса). Для каскадирования таких модулей нужно также иметь возможность подавать на вход перенос от сумматоров младших битов.

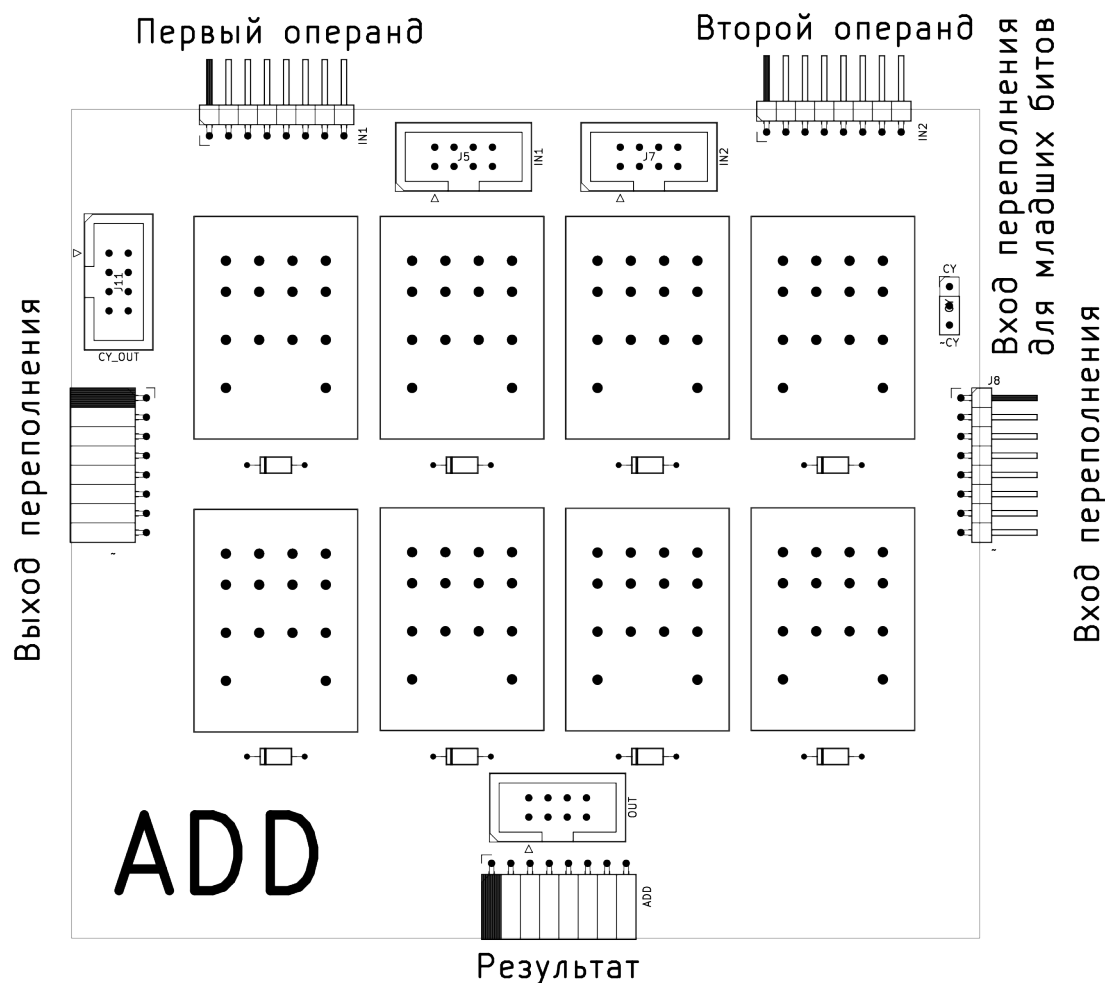
Таким образом, сумматор получает на вход бит переноса и два бита-слагаемых, а на выходе у него тоже бит переноса и однобитная сумма.

Особенность реализации сумматора с помощью реле в том, что на входе требуется не только перенос, но и инвертированный перенос. Такой же инвертированный перенос можно генерировать и на выходе.



Эту схему сумматора изобрёл Конрад Цузе для своих компьютеров в 1940х годах. Несколько таких сумматоров можно соединить последовательно, чтобы складывать многобитные числа. Для этого соединяются соответствующие входы и выходы для переноса и инвертированного переноса, чтобы переполнение переходило из бита 0 в бит 1, из бита 1 в бит 2 и так далее.

## 7.11 Модуль сумматора



Сумматор складывает два четырёхбитных числа, один бит переноса и выдаёт четырёхбитное число и бит переноса.

Модуль имеет следующие разъёмы:

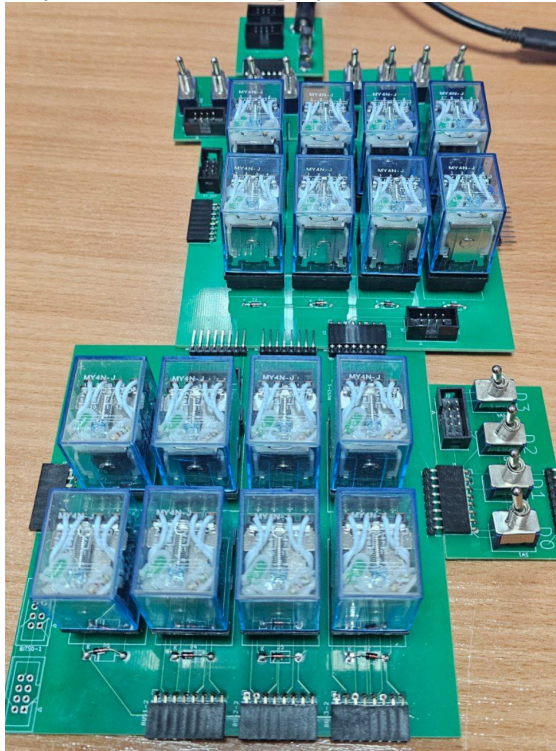
- Слева и справа: шины для каскадирования нескольких модулей. Справа к сумматору приходит сигнал переноса от младших битов (если они есть), а помощью шины слева получившийся в результате сложения перенос можно передать на следующий сумматор, записать в регистр или использовать в других логических схемах.
- Рядом с правым разъёмом есть переключка для включения или выключения бита переноса, если каскадирование не используется. В этом случае переключка обязательно должна быть в одном из двух положений. Если же справа подключен другой сумматор, то переключку нужно убрать.
- Сверху: разъёмы для подключения сигналов-операндов. Можно подсоединить к шинам, а можно к модулям переключателей или регистрам.
- Снизу: разъём, откуда можно считать сумму операндов.

### 7.11.1 Практикум

Список модулей:

- Модуль переключателей: 3 штуки
- Сумматор: 1 штука
- Регистровый модуль: 1 штука

Ко входам модуля подключаются два модуля с тумблерами, а к выходам — регистр, куда будет записываться результат.



1. Отключить все управляющие сигналы.
2. Установить перемычку для подачи сигнала на *СУ*.
3. Набрать на тумблерах первого операнда значение 0011 (число 3).
4. Набрать на тумблерах второго операнда значение 1010 (число 10).
5. Подключить выходной регистр к шине 3. Убедиться, что в него записалось значение 1101 (число 13).
6. Отключить регистр от шины, сбросить его значение.
7. Установить перемычку для подачи сигнала на *СУ*.
8. Подключить выходной регистр к шине 3. Убедиться, что в него записалось значение 1110 (число 14).

### 7.11.2 Задачи

1. Собрать устройство для сложения значения из регистра с константой, набранной на тумблерах. Должна быть предусмотрена запись результата обратно в регистр. Дальше управлять сигналами таким образом, чтобы к регистру последовательно прибавлялась единица, пока он не достигнет значения 1111.

## 7.12 Вычитание

Вычитание в компьютерах делается с помощью трюков с переполнением. Что такое  $10 - 7 = 3$  в нашем четырёхбитном вычислителе? Это  $1010 - 0111 = 0011$ . Так как разрядов всего 4, тот же результат можно получить с помощью сложения. Сначала к 10 прибавим 5, получив 15 ( $1010 + 0101 = 1111$ ). Потом прибавим ещё единицу. Должно было бы получиться 16 (10000), но так как значащих битов всего 4, пятый бит результата пропадёт и останется значение 0 (0000).

Остаётся увеличить это число на единицу ещё три раза, чтобы получился правильный ответ. Итого, мы прибавили к 10 число 9 вместо вычитания 5 и получили нужный результат.

Следовательно, вместо вычитания 7 (0111) можно прибавлять 9 (1001), когда речь идёт о четырёхбитных вычислениях.

Такой же фокус можно проделать с любым числом. Быстрый алгоритм работает так: инвертируем все биты числа, а затем прибавляем единицу. Это называется дополнительным обратным кодом.

Например, для 0111 сначала получится 1000, а потом 1001, что и требовалось. Работает и в обратную сторону — из 1001 инверсией получаем 0110, а после добавления единицы будет 0111.

Таким образом, для вычитания не нужно строить сложную схему, а достаточно использовать дополнительный обратный код для второго операнда при сложении.

## 7.13 Вычитание через сложение с дополнительным обратным кодом

Чтобы вычесть из одного числа другое, можно вычитаемое представить в дополнительном обратном коде, а затем сложить это число с уменьшаемым.

Число в дополнительном обратном коде получается, если сначала инвертировать биты исходного числа, а затем прибавить к нему единицу.

Например, для числа  $3 = 0011$  инверсия будет выглядеть, как 1100, а дополнительный обратный код, как 1101.

### Практикум

Список модулей:

- Модуль переключателей: 3 штуки
- Сумматор: 1 штука
- Регистровый модуль: 1 штука

Собрать схему для сложения.

1. Отключить все управляющие сигналы.
2. Установить переключку для подачи сигнала на  $CY$ .
3. Набрать на тумблерах первого операнда значение 1010 (число 10).
4. Набрать на тумблерах второго операнда значение 1101 (число  $-3$ ).
5. Подключить выходной регистр к шине 3. Убедиться, что в него записалось значение 0111 (число 7).

### Задачи

1. Выполните деление в столбик числа 14 на 3 с помощью последовательности вычитаний. Записывайте промежуточные результаты на бумаге.
2. Сделайте то же самое, только сохраняйте все промежуточные результаты в разных регистрах регистрового файла.

## 7.14 Вычитание с помощью сумматора и инвертора

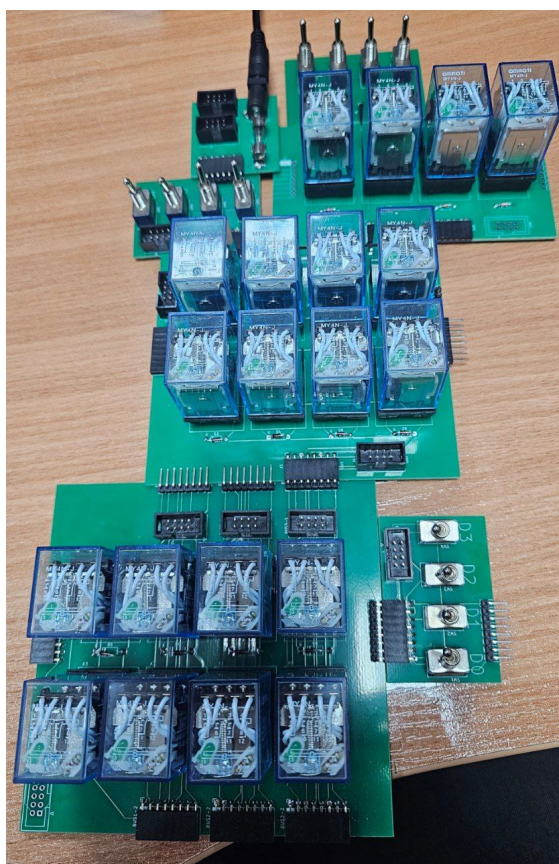
Вычитание можно делать с помощью сумматора. Чтобы на входе из операнда получался дополнительный обратный код, сначала нужно преобразовать его с помощью инвертора, а затем прибавить 1, включив вход переноса в сумматоре.

### Практикум

Список модулей:

- Модуль переключателей: 3 штуки
- Модуль унарных операций: 1 штука
- Сумматор: 1 штука
- Регистровый модуль: 1 штука

Собрать схему для сложения, а затем добавить между тумблерами со вторым операндом инвертор (выход NOT блока унарной логики):



1. Отключить все управляющие сигналы.
2. Установить перемычку для подачи сигнала на  $CY$ .
3. Набрать на тумблерах первого операнда значение 1010 (число 10).
4. Набрать на тумблерах второго операнда значение 0011 (число 3).
5. Подключить выходной регистр к шине 3. Убедиться, что в него записалось значение 0111 (число 7).

## Глава 8

# Калькулятор

### 8.1 Чем калькулятор отличается от компьютера?

### 8.2 Калькулятор для 7 операций

Все вычислители можно соединить с одним и тем же регистром, как хранилищем результата, чтобы получить простейший калькулятор. Доступны следующие операции:

1. Инверсия
2. Сдвиг вправо
3. Побитовое И
4. Побитовое ИЛИ
5. Исключающее ИЛИ
6. Сложение
7. Вычитание

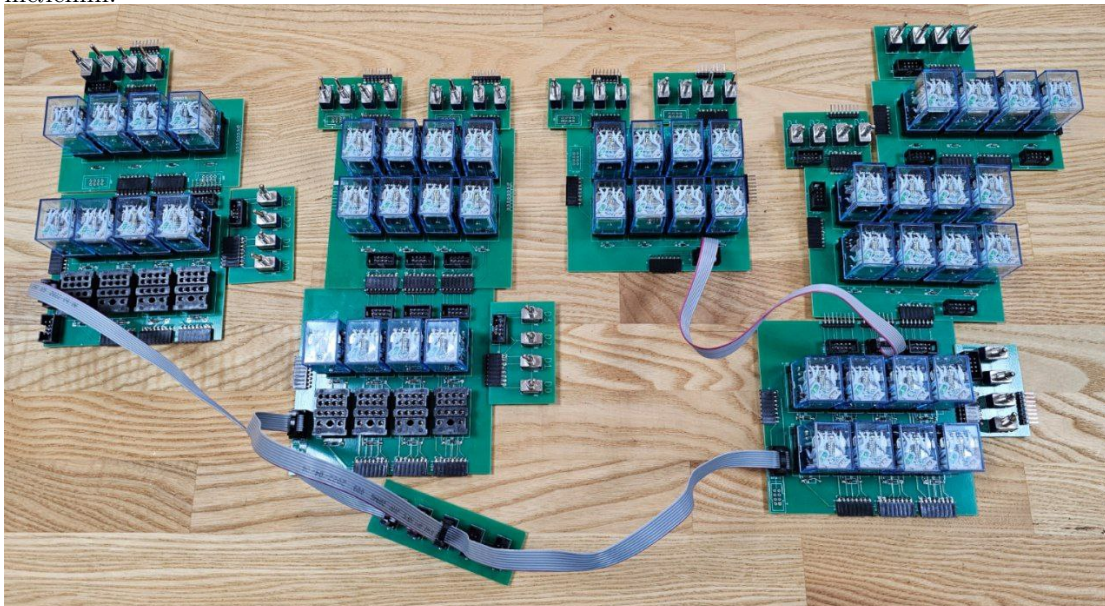
#### Практикум

Список модулей:

- Модуль переключателей: 10 штук
- Модуль унарных операций: 2 штуки
- Сумматор: 2 штуки
- Модуль логических операций: 1 штука
- Регистровый модуль как шинный формирователь: 2 штуки
- Регистровый модуль: 1 штука
- Модуль шины: 1 штука
- Соединительные шлейфы: 4 штуки



Уже проверенные модули для разных операций соединяются вместе. Для этого используются несколько мультиплексоров (шинных формирователей) на платах регистров. На этих платах не установлены реле для хранения битов. Вместо этого шины подключаются к одному и тому же регистру. Так в него можно записывать любой из 7 результатов вычислений.



Любую из операций можно выполнить так:

1. Отключить все управляющие сигналы.
2. Набрать входные данные для нужной операции.
3. Подключить выход нужного модуля к регистру тумблером.
4. Наблюдать результаты вычислений.
5. Сбросить значение регистра.

### 8.3 Расширение вычислений до восьми бит

У регистров и вычислительных модулей справа и слева есть разъёмы для расширения разрядности.

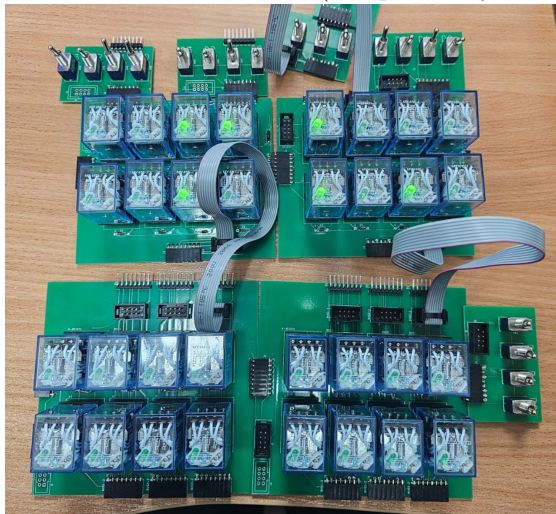
Для регистров через эти разъёмы передаются сигналы сброса и подключения к шинам. Поэтому один набор тумблеров может использоваться для двух четырёхбитных плат-регистров, если они соединены в один восьмибитный регистр.

Для вычислительных модулей через боковые разъёмы передаются сигналы переноса (в случае сдвига и сложения).

#### Практикум

Соединить два регистра и два сумматора. Шина 3 регистра подключается к сумматору. У правого (младшего) сумматора входящий перенос перемычкой устанавливается в 0. У лево-

го (старшего) сумматора переключатель для переноса убирается, потому что сигнал переноса приходит от младших битов (из правого сумматора).



1. Отключить все управляющие сигналы.
2. Набрать входные данные 00111001 и 00101001.
3. Подключить выход сумматора к регистру тумблером  $D3$ .
4. Наблюдать результат вычислений 01100010.

## Часть III

# Строим релейный компьютер

## Глава 9

# Элементы компьютера

### 9.1 Дешифратор

#### 9.1.1 Практикум

### 9.2 ПЗУ

### 9.3 Набор инструкций

### 9.4 Декодер инструкций

### 9.5 Тактовый генератор

## Глава 10

# Компьютер