



## ЛАБОРАТОРНАЯ РАБОТА 1

### Программирование ПЛК на языке LD.

**Цель работы:** Изучить принципы составления прикладных программ для промышленных логических контроллеров на языке LD пакета CoDeSys.

#### Общие сведения.

Язык LD (Ladder Diagram) или РКС (Релейно-Контактные Схемы) представляет собой графическую форму записи логических выражений в виде контактов и обмоток реле.

LD предназначен для программирования промышленных контроллеров (ПЛК). Синтаксис языка удобен для замены логических схем, выполненных на релейной технике.

Слева и справа схема ограничена вертикальными линиями - шинами питания. Между ними расположены цепи, образованные контактами и обмотками реле, по аналогии с обычными электронными цепями. Слева любая цепь начинается набором контактов, которые посылают слева направо состояние «ON» или «OFF», соответствующие логическим значениям ИСТИНА или ЛОЖЬ. Каждому контакту соответствует логическая переменная. Если переменная имеет значение ИСТИНА, то состояние передается через контакт. Иначе правое соединение получает значение выключено ("OFF").

Контакты могут быть соединены параллельно, тогда соединение передает состояние логическое ИЛИ. Если контакты соединены последовательно, то соединение передаёт логическое И.

Контакт может быть инвертируемым. Такой контакт обозначается с помощью символа  $|/|$  и передает состояние "ON", если значение переменной ЛОЖЬ.



LD позволяет:

- выполнять последовательное соединение контактов;
- выполнять параллельное соединение контактов;
- применять нормально разомкнутые или замкнутые контакты;
- использовать переключаемые контакты;
- записывать комментарии;
- включать Set / Reset-выходы;
- переходы;
- включать в диаграмму функциональные блоки;
- управлять работой блоков по входам EN.

### **Порядок выполнения работы**

Из ПРИЛОЖЕНИЯ 1 по заданию преподавателя выбирается вариант технологической схемы.

На основании заданной технологической схемы и описания технологического процесса разработать:

- технологические требования к схеме управления;
- принципиальную электрическую схему автоматического управления технологической установкой;
- прикладную программу для ПЛК.
- дать описание работы принципиальной схемы.



Схема и программа должна предусматривать:

- запуск всех машин и механизмов в последовательности, направленной против движения продукта;
- остановку всех машин и механизмов в последовательности, совпадающей с направлением движения продукта;
- остановку поточных линий по команде «рабочий стоп» с целью очистки тракта;
- режим пуско-наладочных работ;
- звуковой или световой сигнал при пуске сложных технологических установок;
- аварийное отключение (при аварийном отключении одной из машин, должны остановиться без выдержки времени все машины, работающие на ее загрузку, а с выдержкой времени все машины работающие на отгрузку).

### **Составление прикладной программы.**

1. При создании программы используется среда программирования **CoDeSys V2.3** (далее – **CoDeSys**).

Перед созданием проекта пользователь, используя утилиту **InstallTarget** в составе **CoDeSys**, устанавливает для применяемого контроллера файл целевых задачи (**Target, файл**), который обеспечивает программный доступ к ресурсам ПЛК.

### **2. Создание проекта программы**

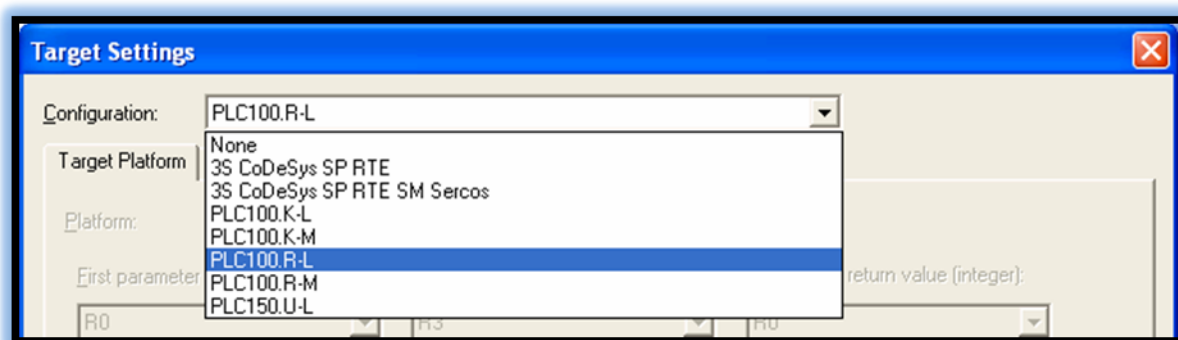
При создании проекта используется язык релейных диаграмм **LD (Ladder Diagram)**, реализующий структуры, подобные электрическим цепям в коммутационной автоматике.



Пользователь запускает **CoDeSys** последовательным выбором приложений:

**Пуск → Все программы → 3S Software → CoDeSys V2.3 → CoDeSys V2.3.**

Новый проект открывается из главного меню: **File → New**. В открывшемся окне (рис .1) выбирается тип контроллера, **PLC150.U-L**, выбор подтверждается нажатием клавиши **OK**.



**Рисунок 1.1- Окно конфигурации «Target Settings» программы**

После выбора проекта выводится экранная форма, задающая тип, имя и язык программирования первичного компонента **New POU**, главной программы контроллера. Необходимо выбрать язык программирования **LD**, установив флаги в позициях, указанных на рис.1.2.

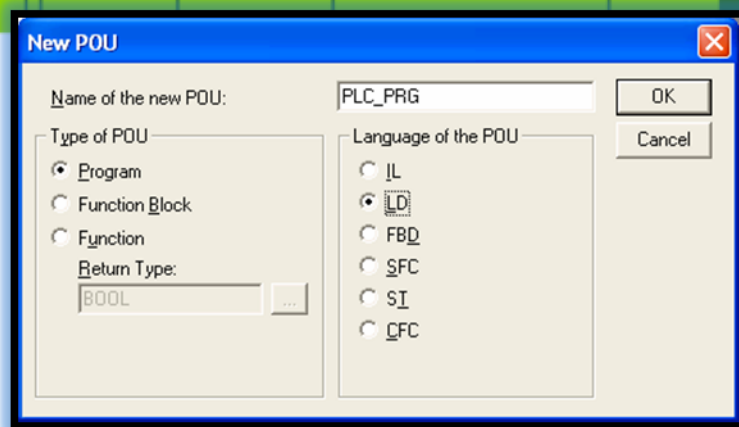


Рисунок 1.2- Вид окна «New POU» с отмеченными параметрами

***Примечание.** Имя главной программы PLC\_PRG и ее тип менять нельзя.*

После подтверждения выбора нажатием клавиши **ОК** откроется окно нового проекта с именем по умолчанию **Untitled**. В нем присутствует одна вкладка **POUs**.

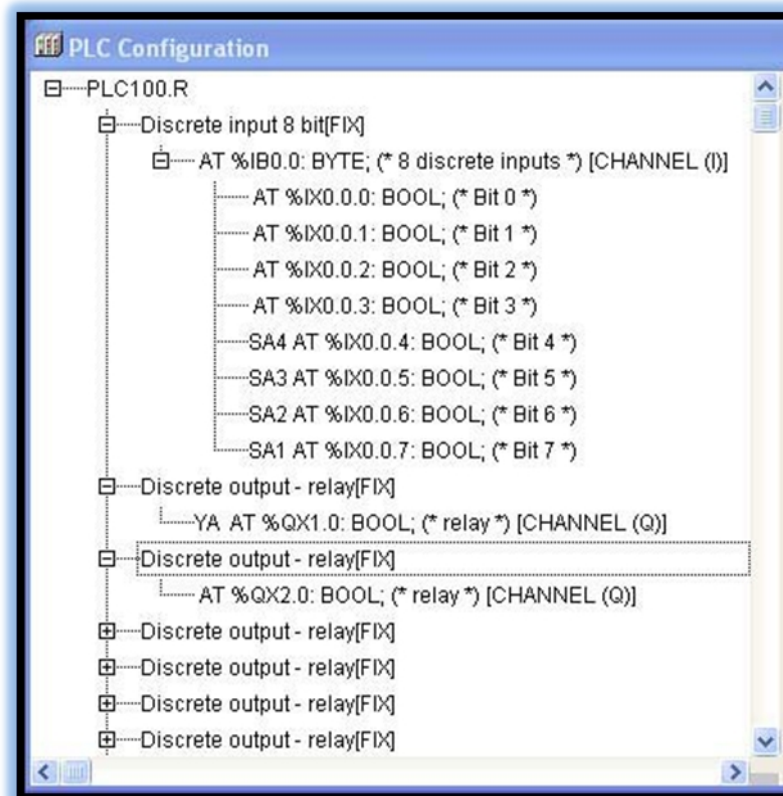
Весь проект хранится в одном файле, имя которого отображается в заголовке окна. Для ввода имени файла во второй строке меню быстрого запуска активизируется клавиша записи и в появившейся форме указывается имя файла: например, **LD, проект 1.pro**.

### 3. Параметры входов и выходов контроллера

Цепям контроллера, используемым в разрабатываемой электрической схеме, присваиваются имена переменных. В дальнейшем эти имена используются в программе для работы с конкретным входом или выходом контроллера.

Для присвоения имени какому-либо ресурсу ввода/вывода контроллера необходимо на вкладке ресурсов (**Resources**) Организатора объектов **CoDeSys** запустить утилиту **PLC Configuration (Конфигуратор ПЛК)**.

ПЛК –



пользователь открывает папки (модули) входов (**Discrete input**) и выходов (**Discrete output**) ПЛК, и именуется необходимые каналы. Перед адресом указывается имя (идентификатор переменной) для цепей входов и выходов схемы созданного проекта.

Именованье канала (входа или выхода) производится следующим образом: двойным щелчком манипулятора «мышь» при курсоре, установленном в начале строки названия канала, осуществляется переход в режим редактирования и вводится имя переменной канала.

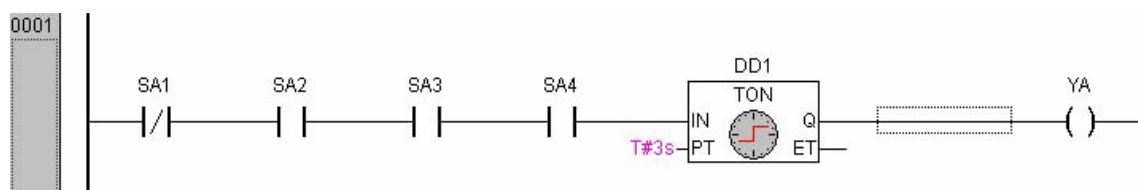
Экранная форма, представленная иллюстрируют выполненные пользователем именования каналов при использовании четырех входов (**IX0.0.4, IX0.0.5, IX0.0.6, IX0.0.7**) и одного выхода (**QX1.0**).



**Рисунок 1.3- Экранная форма для именования входов и выходов  
при работе программы с цепями ПЛК**

#### **4. Создание программы на языке LD**

При написании программы в рабочей зоне вкладки **POUs** последовательно вводятся типы компонентов и их обозначения, как это представлено на рис.1. 4.




## Рисунок 1.4. Пример программы на языке LD

Пользователь при составлении виртуальной схемы может следовать приведенной ниже инструкции.




### Инструкция по созданию программы:

1) **создание нормально замкнутого контакта:** в контекстном меню выбрать команду **Contact (negated)** или нажать кнопку  на панели инструментов. Символы вопросов (рис. 1.5 (а)) необходимо заменить именем, например **SA1**.

Описывать переменную в данном случае не требуется, так как она уже была указана в окне PLC(Configuration и связана с конкретным дискретным входом;



Рисунок 1.5 - Создание нормально замкнутого (а)  
и разомкнутого (б) контактов

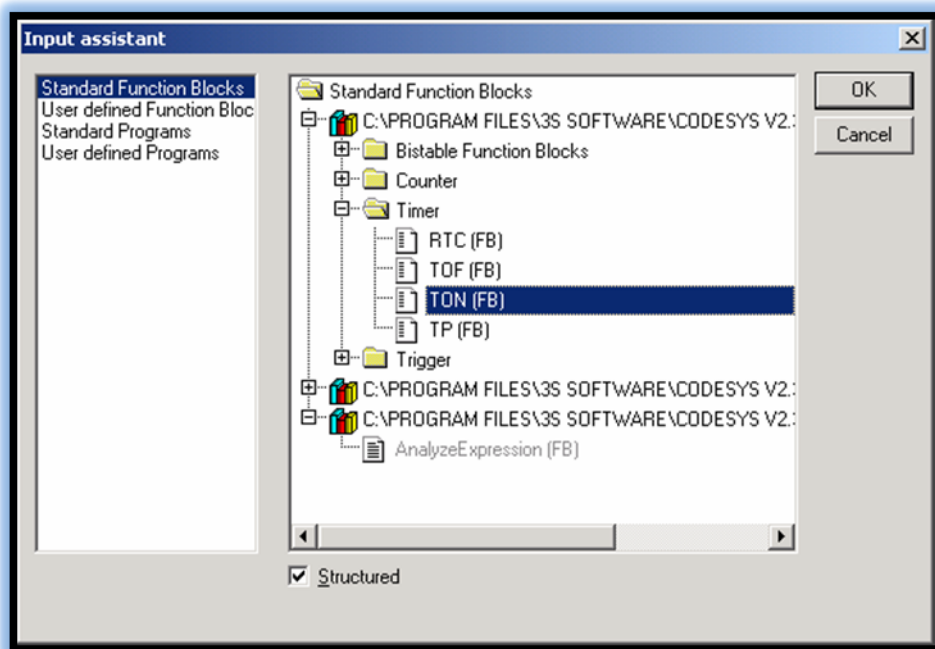
2) **создание нормально разомкнутого контакта** делается аналогичным образом, только используется команда контекстного меню **Contact** или кнопка  на панели инструментов (рис.1.5(б));

3) **функциональный блок:** из контекстного меню выбирается команда **Function Block...**, – в появившемся окне «**Input Assistant**» (рис.1.6) из раздела **Standard Function Blocks** в библиотеке с именем


**STANDARD.LIB** в папке **Timer** выбирается вид таймера – **TON (FB)**. На схеме перед входом **PT** указывается время задержки



в формате **T#3s**. Над блоком вводится имя, например **DD1** и на клавиатуре нажимается клавиша «стрелка вправо» подтверждаются свойства функционального блока;



**Рисунок 1.6 - Выбор таймера**

5) **указание выхода цепи:** в контекстном меню выбирается команда **Coil** или нажимается кнопка  на панели инструментов. На схеме появляется условное обозначение обмотки реле. Символы вопросов замещаются именем.

## **6. Запись программы в контроллер**

Настройка соединения ПК с ОВЕН ПЛК для загрузки и проверки работы программы в автономном режиме производится следующим образом.

Для информационного обмена ПК с ОВЕН ПЛК используется кабель

программирования, входящий в комплект поставки. Им соединяются **COM**(порт компьютера и порт **Debug RS,232** контроллера (на лицевой панели)).

Для настройки канала соединения из основного меню **CoDeSys** выбирается команда **Online-Communication parameters**. В диалоговом меню командой **New...** открывается диалоговое окно, в котором соединению присваивается имя (например, **COM**) и выбирается (из перечня) вид соединения **Serial (RS232)**. Выбор подтверждается нажатием клавиши **OK**.



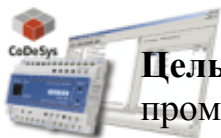
После указанных действий в окне коммуникационных параметров появляется канал соединения с именем **COM**. В зоне настроек (**Value**) для параметра **Baudrate** устанавливается значение **115200** (бит/сек – скорость соединения с компьютером). Значение может быть изменено двойным щелчком левой кнопки манипулятора «мышь» на значении. Для сохранения нового значения в окне курсором мыши выбор подтверждается нажатием клавиши **OK**.

Программное соединение с ОВЕН ПЛК включается из главного меню **CoDeSys** командой **Online - Login**. При этом флаг перед строкой меню **Simulation Mode** должен быть снят.

Как только система устанавливает связь с ОВЕН ПЛК, появляется запрос на подтверждение загрузки новой программы, пользователь подтверждает загрузку: **Да**.

После завершения записи проекта в оперативную память ОВЕН ПЛК, запуск работы программы осуществляется выбором команды **Online - Run** (или нажатием на лицевой панели ОВЕН ПЛК кнопки **<Старт>**). Управлять работой ОВЕН ПЛК можно при помощи переключателей.

## ЛАБОРАТОРНАЯ РАБОТА 2



**Цель работы:** Изучить принципы составления прикладных программ для промышленных логических контроллеров на языке ST пакета CoDeSys.

### 1. Общие сведения.

Язык ST (Structured Text) — это язык высокого уровня. Синтаксически ST представляет собой несколько адаптированный язык Паскаль. Вместо процедур Паскаля в ST используются компоненты программ стандарта МЭК. На основе ST можно создавать гибкие процедуры обработки данных.

Основой ST-программы служат выражения. Результат вычисления выражения присваивается переменной при помощи оператора «:=», как и в Паскале. Каждое выражение обязательно заканчивается точкой с запятой «;». Выражение состоит из переменных констант и функций, разделенных операторами:

```
iVar1 := 1 + iVar2 / ABS(iVar2);
```

Стандартные операторы в выражениях ST имеют символьное представление, например математические действия: +, -, \*, / операции сравнения и т.д.

Помимо операторов, элементы выражения можно отделять пробелами и табуляциями для лучшего восприятия.

Таблица 1 - Операторы в ST

Операция	Обозначение	Приори
Выражение в скобках	(Выражение)	Самый высокий
	Имя функции (список параметров)	
	EXPT	
	-	
	NOT	
	*	
	/	
	MOD	
	+	
	-	
	<, >, <=, >=	
	< >	
	=	
	AND	
	XOR	
	OR	Самый низкий



## Основные типовые конструкции в ST

### Оператор присваивания

Перед оператором присваивания находится операнд (переменная или адрес), которому присваивается значение выражения, стоящего после оператора присваивания.


*Пример:*

Var1: = Var2 \* 10;

После выполнения этой операции Var1 принимает значение в десять раз большее, чем Var2.

### Вызов функционального блока в ST

Функциональный блок вызывается с помощью имени экземпляра функционального блока и списка входных параметров с присваиванием



данных в круглых скобках. В следующем примере вызывается таймер с параметрами IN и PT. Значение выходной переменной Q присваивается переменной A. К выходной переменной можно обратиться с помощью имени экземпляра функционального блока, точки, следующей за ним и имени выходной переменной:

```
CMD_TMR (IN: = %IX5, PT: = 300);  
A := CMD_TMR Q
```

### Оператор выбора IF

*Оператор выбора* позволяет выполнить различные группы выражений в зависимости от условий, выраженных логическими выражениями. Полный синтаксис оператора IF (если) выглядит так:

```
IF <логическое выражение IF>  
THEN  
  <выражения IF>  
  [  
    ELSIF <логическое выражение ELSEIF 1>  
    THEN  
      < выражения ELSEIF 1> ;  
    ...  
    ELSIF <логическое выражение ELSEIF n>  
    THEN  
      < выражения ELSEIF n> ;  
    ELSE  
      < выражения ELSE> ;  
  ]  
END IF
```

Если <логическое выражение IF> ИСТИНА, то выполняются выражения первой группы - <выражения IF>. Прочие выражения пропускаются, альтернативные условия не проверяются. Часть

конструкции в квадратных скобках является необязательной и может отсутствовать. Если <логическое выражение IF> ЛОЖЬ, то одно за другим проверяются условия ELSIF. Первое истинное условие приведет к

выполнению соответствующей группы выражений. Прочие условия **ELSIF** анализироваться не будут. Групп **ELSIF** может быть несколько или не быть совсем. Если все логические выражения дали ложный результат, то выполняются выражения группы **ELSE**, если, она есть. Если группы **ELSE** нет, то не выполняется ничего. В простейшем случае оператор **IF** содержит только одно условие:

```
IF bReset THEN
```

```
    iVar1 := 1;
```

```
    iVar2 := 0;
```

```
END_IF
```

### **Оператор множественного выбора CASE**

*Оператор множественного выбора CASE* позволяет выполнить различные группы выражений в зависимости от значения одной целочисленной переменной или выражения. Синтаксис:

```
CASE <целочисленное выражение> OF
```

```
    <значение 1> :
```

```
    <выражения 1> ;
```

```
    <значение 2> , значение 3> :
```

```
        <выражения 3> ;
```

```
    <значение 4>..значение 5> :
```

```
        <выражения 4> ;
```

```
    ...
```

```
    [
```

```
    ELSE
```

```
        <выражения ELSE>;
```

```
    ]
```

```
END CASE
```

Если значение выражения совпадает с заданной константой, то выполняется соответствующая группа выражений. Прочие условия не

анализируются (<значение 1>: <выражения 1> ;). Если несколько

значений констант должны соответствовать

одной группе выражений, их можно перечислить через запятую (<значение 2> , <значение3> : <выражения 3> ;). Диапазон значений можно определить через две точки (<значение 4>..<>значение 5> : <выражения 4> ;). Группа выражений ELSE является необязательной. Она выполняется при несовпадении ни одного из условий (<выраженияELSE> ;).

Пример:

**CASE byLeft/2 OF**

**0,127:**

bReset := TRUE;

Var1 :=0;

**16..24:**

Var1 :- 1;

**ELSE**

Var1 := 2;

**END\_CASE**

Значениями выбора **CASE** могут быть только целые константы, переменные использовать нельзя. Одинаковые значения в альтернативах выбора задавать нельзя, даже в диапазонах.

## Цикл FOR

С помощью FOR можно программировать повторяющиеся процессы. Синтаксис:

```
FOR <Целый счетчик> := <Начальное
значение>
TO <Конечное значение>
[BY <Шаг>] DO
<Выражения - тело цикла>
```



Перед выполнением цикла счетчик получает начальное значение. Далее тело цикла повторяется, пока значение счетчика не превысит конечного значения. Счетчик увеличивается в каждом цикле. Начальное и конечное значения и шаг могут быть как константами, так и выражениями. Счетчик изменяется после выполнения тела цикла. Поэтому если задать конечное значение меньше начального, то при положительном приращении цикл не будет выполнен ни разу. При одинаковых начальном и конечном значениях тело цикла будет выполнено один раз. Часть конструкции ВУ в скобках необязательна, она определяет шаг приращения счетчика. По умолчанию счетчик увеличивается на единицу в каждой итерации. В качестве счетчика можно использовать переменную любого целого типа.

Пример:

Varl := 0;

**FOR** cw := 1 TO 10 **DO**

Varl := Varl + 1;

**END\_FOR**

Данный цикл будет выполнен 10 раз и соответственно Varl будет иметь значение 10.

Шаг изменения счетчика итераций может быть и отрицательным. Начальное условие в этом случае должно быть больше конечного. Цикл будет закончен, когда значение счетчика станет меньше конечного значения.

Цикл FOR исключительно удобен для итераций с заранее известным числом повторов.

Для построения правильного цикла достаточно соблюдать два



простых формальных требования:

- не изменяйте счетчик цикла и условие окончания в теле цикла.

Счетчик и переменные образующие конечное условие в цикле, можно использовать только для чтения;

- не задавайте в качестве конечного условия максимальное для типа переменной счетчика значение. Так, если для однобайтного целого без знака задать константу 255, то условие окончания не будет выполнено никогда. Цикл станет бесконечным.

### Циклы WHILE и REPEAT

Циклы **WHILE** и **REPEAT** обеспечивают повторение группы выражений, пока верно условное логическое выражение. Если условное выражение всегда истинно, то цикл становится бесконечным.

#### Синтаксис WHILE:

```
WHILE <Условное логическое выражение> DO  
    <Выражения — тело цикла>  
END WHILE
```

Условие в цикле **WHILE** проверяется до начала цикла. Если логическое выражение изначально имеет значение **ЛОЖЬ**, тело цикла не будет выполнено ни разу.

Пример:

```
ci := 64;  
WHILE ci > 1 DO  
    Varl := Varl + 1;  
    ci := ci/2;  
END WHILE
```

#### Синтаксис REPEAT:

```
REPEAT  
    <Выражения - тело цикла >  
UNTIL <Условное логическое выражение>  
END REPEAT
```

Правильно построенный цикл WHILE или REPEAT обязательно должен изменять переменные, составляющие условие окончания в теле цикла, постепенно приближаясь к условию завершения. Если этого не сделать, цикл не закончится никогда.



## 2. Порядок выполнения работы

Из ПРИЛОЖЕНИЯ 2 по заданию преподавателя выбирается вариант технологической схемы.

На основании заданной технологической схемы и описания технологического процесса разработать:

- технологические требования к схеме управления;
- таблицу сигналов;
- прикладную программу для ПЛК.
- дать описание работы принципиальной схемы.

В таблицу сигналов вносятся:

- порядковый номер переменной;
- имя переменной (не должно содержать пробелов и кириллицы);
- тип переменной (дискретный, аналоговый);
- класс переменной (локальная, глобальная);
- адрес (для внутренних переменных не заполняется).

**Пример заполнения таблицы параметров.**

№	Наименование	Имя	Тип	Класс	Адрес
---	--------------	-----	-----	-------	-------

п/п	параметра				
1.	Температура воздуха	Temp_1	аналог	глобал.	%IB0
2.	Кнопка «Пуск»	SB1	дискр.	локал.	-

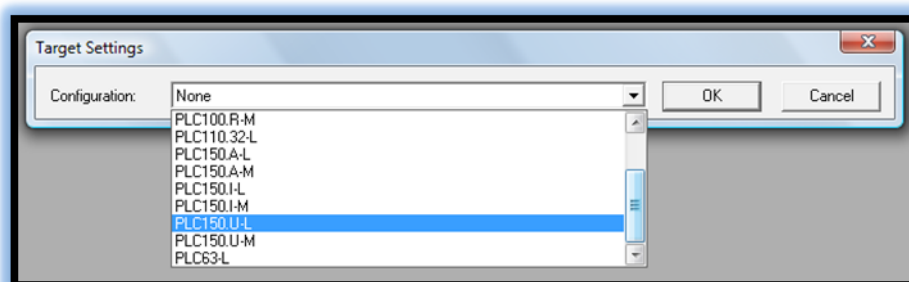
### 3. Создание проекта программы.

При создании проекта используется язык структурированного текста **ST**, реализующий структуры, подобные структурам языка Pascal.

Пользователь запускает **CoDeSys** последовательным выбором приложений:

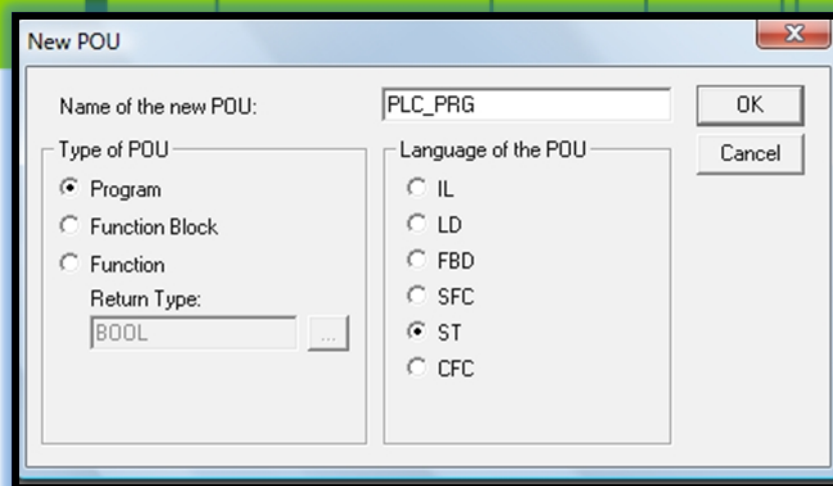
**Пуск → Все программы → 3S Software → CoDeSys V2.3 → CoDeSys V2.3.**

Новый проект открывается из главного меню: **File → New**. В открывшемся окне (рис. 2.1) выбирается тип контроллера, **PLC150.U-L**, выбор подтверждается нажатием клавиши **OK**.



**Рисунок 2.1-Окно конфигурации «Target Settings» программы**

После выбора проекта выводится экранная форма, задающая тип, имя и язык программирования первичного компонента **New POU**, главной программы контроллера. Необходимо выбрать язык программирования **ST**, установив флаги в позициях, указанных на рис.2.



**Рисунок 2.2- Вид окна «New POU» с отмеченными параметрами**

***Примечание.** Имя главной программы PLC\_PRG и ее тип менять нельзя.*

После подтверждения выбора нажатием клавиши **OK** откроется окно нового проекта с именем по умолчанию **Untitled**.

#### **4. Параметры входов и выходов.**

Параметры входов и выходов контроллера задаются на вкладке ресурсов (**Resources**) Организатора объектов **CoDeSys** с помощью утилиты **PLC Configuration (Конфигуратор ПЛК)** (LP№1 п.3).

#### **5. Создание программы на языке ST**

При написании программы в рабочей зоне вкладки **POUs** вводится текст, состоящий из имен переменных и операторов (Таблица 1). При необходимости используются приведенные выше конструкции IF, FOR, CASE, WHILE и REPEAT.

В тексте программы необходимо учесть технологические требования, предъявляемые к системе управления.



## ЛАБОРАТОРНАЯ РАБОТА 3

### Программирование ПЛК на языке IL

**Цель работы:** Изучить принципы составления прикладных программ для промышленных логических контроллеров на языке IL пакета CoDeSys.

#### 1. Общие сведения

Язык IL (Instruction list) дословно — список инструкций.

Это типичный ассемблер с аккумулятором и переходами по меткам. Набор инструкций стандартизован и не зависит от конкретной целевой платформы. Поскольку IL самый простой в реализации язык, он получил очень широкое распространение.

Наибольшее влияние на формирование современного IL оказал язык программирования STEP контроллеров фирмы Siemens. Язык IL позволяет работать с любыми типами данных, вызывать функции и функциональные блоки, реализованные на любом языке. Таким образом, на IL можно реализовать алгоритм любой сложности, хотя текст будет достаточно громоздким.

В составе МЭК-языков IL применяется при создании компактных компонентов, требующих тщательной проработки, на которую не жалко времени. При работе с IL гораздо адекватнее, чем с другими языками, можно представить, как будет выглядеть оттранслированный код. Благодаря чему, IL выигрывает там, где нужно достичь наивысшей эффективности. К компиляторам это относится в полной мере. В системах исполнения с интерпретатором промежуточного кода выигрыш не столь значителен.

Текст на IL — это текстовый список последовательных инструкций. Каждая инструкция записывается на отдельной строке. Инструкция может включать 4 поля, разделенные пробелами или знаками табуляции:

Метка:	Оператор	Операнд	Комментарий
--------	----------	---------	-------------



Метка инструкции не является обязательной, она ставится только там, где нужно. Оператор присутствует обязательно. Операнд необходим почти всегда. Комментарий - необязательное поле, записывается в конце строки. Ставить комментарии между полями инструкции нельзя. Пример IL-программы:

METKA1:	<b>LD</b>	<b>Sync</b>	(*пример IL*)
	<b>AND</b>	<b>Start</b>	
	<b>S</b>	<b>Q</b>	

Для лучшего восприятия строки IL выравнивают обычно в колонки по полям. Редактор CoDeSys выравнивает текст автоматически. Помимо этого, редактор «налету» выполняет синтаксический контроль и выделение цветом. Так, корректно введенные операторы выделяются голубым цветом.

### Аккумулятор

Абсолютное большинство инструкции IL выполняют некоторую операцию с содержимым аккумулятора. Операнд, конечно, тоже принимает участие в инструкции, но результат опять помещается в аккумулятор. Например, инструкция SUB 10 отнимает число 10 от значения аккумулятора и помещает результат в аккумулятор. Команды сравнения сравнивают значение операнда и аккумулятора, результат сравнения ИСТИНА или ЛОЖЬ вновь помещается в аккумулятор. Команды перехода на метку способны анализировать аккумулятор и принимать решение - выполнять переход или нет.

Аккумулятор IL является универсальным контейнером, способным сохранять значения переменных любого типа.

В аккумулятор можно поместить значение типа **BOOL**, затем



**INT** или **REAL**, транслятор не будет считать это ошибкой. Такая гибкость не означает, что аккумулятор способен одновременно содержать несколько значений разных типов. Только одно, причем тип значения также фиксируется в аккумуляторе. Если операция требует значение другого типа, транслятор выдаст ошибку. В стандарте МЭК вместо термина «аккумулятор» используется термин «результат» (result). Так, инструкция берет «текущий результат» и формирует «новый результат». Тем не менее почти все руководства по программированию различных фирм широко используют термин «аккумулятор».

### Переход на метку

Программа на IL выполняется подряд, сверху вниз. Для изменения порядка выполнения и организации циклов применяется *переход на метку*. Переход на метку может быть безусловным **JMP** - выполняется всегда, независимо от чего-либо. Условный переход **JMPC** выполняется только при значении аккумулятора ИСТИНА. Переход можно выполнять как вверх, так и вниз. Метки являются локальными, другими словами, переход на метку в другом POU не допускается.

Переходы нужно организовывать достаточно аккуратно, чтобы не получить бесконечный цикл:

### Скобки

Последовательный порядок выполнения команд IL можно изменять при помощи *скобок*. Открывающая скобка ставится в инструкции после операции. Закрывающая скобка ставится в отдельной строке. Инструкции, заключенные в скобки, выполняются в первую очередь. Результат вычисления инструкций в скобках помещается в дополнительный

аккумулятор, после чего выполняется команда, содержащая открывающую скобку. Например:



```
LD      1
ST      Counter
LD      5
MUL     (2
SUB     1
)
ST      y      (*y = 5 * (2-1) = 5*)

LD      5
MUL     2
SUB     1
ST      y      (*y = 5 * 2 - 1 = 9*)
```

Скобки могут быть вложенными. Каждое вложение требует организации некоего временного аккумулятора. Это вызывает неоднозначность при выходе из блока скобок командами JMP, RET, CAL и LD. Применять эти команды в скобках нельзя.

### Модификаторы

Добавление к мнемонике некоторых операторов символов - модификаторов «C» и «N» модифицирует смысл инструкции. Символ «N» (negation) вызывает диверсию значения операнда до выполнения инструкции. Операнд должен быть типов **BOOL**, **BYTE**, **WORD** или **DWORD**.

Символ «C» (condition) добавляет проверку условий к командам перехода, вызова и возврата. Команды **JMPC**, **CALC**, **RETC** будут выполняться только при значении аккумулятора ИСТИНА. Добавление символа 'N' приводит к сравнению условия с инверсным значением аккумулятора. Команды **JMPCN**, **CALCN**, **RETCN** будут выполняться только при значении аккумулятора ЛОЖЬ. Модификатор «N» без «C» не имеет смысла в данных операциях и не применяется.



## Операторы



Стандартные операторы IL с допустимыми модификаторами представлены в таблице:

Оператор	Модификатор	Описание
<b>LD</b>	N	Загрузить значение операнда в аккумулятор
<b>ST</b>	N	Присвоить значение аккумулятора операнду
<b>S</b>		Если аккумулятор ИСТИНА, установить логический операнд (ИСТИНА)
<b>R</b>		Если аккумулятор ИСТИНА, сбросить логический операнд (ЛОЖЬ)
<b>AND</b>	N, (	Поразрядное И
<b>OR</b>	N, (	Поразрядное ИЛИ
<b>XOR</b>	N, (	Поразрядное ИЛИ
<b>NOT</b>		Поразрядная инверсия аккумулятора
<b>ADD</b>	(	Сложение
<b>SUB</b>	(	Вычитание
<b>MUL</b>	(	Умножение
<b>DIV</b>	(	Деление
<b>MOD</b>	(	Деление по модулю
<b>GT</b>	(	>
<b>GE</b>	(	=>
<b>QE</b>	(	=
<b>NE</b>	(	< >
<b>LE</b>	(	<=
<b>LT</b>	(	<
<b>JMP</b>	CN	Переход к метке
<b>CAL</b>	CN	Вызов функционального блока
<b>RET</b>	CN	Выход из ROU и возврат в вызывающую программу.

Операторы S и R применяются только с операндами типа **BOOL**. Прочие операторы работают с любыми переменными базовых типов.

Приведенный список содержит операторы, поддерживаемые в обязательном порядке. Трансляторы кода CoDeSys для различных

аппаратных платформ реализуют различные подмножества дополнительных операторов.



## 2. Порядок выполнения работы

Из ПРИЛОЖЕНИЯ 2 по заданию преподавателя выбирается вариант технологической схемы.

На основании заданной технологической схемы и описания технологического процесса разработать:

- технологические требования к схеме управления;
- таблицу сигналов;
- прикладную программу для ПЛК.
- дать описание работы прикладной программы.

В таблицу сигналов вносятся:

- порядковый номер переменной;
- имя переменной (не должно содержать пробелов и кириллицы);
- тип переменной (дискретный, аналоговый);
- класс переменной (локальная, глобальная);
- адрес (для внутренних переменных не заполняется).

**Пример заполнения таблицы параметров.**

№ п/п	Наименование параметра	Имя	Тип	Класс	Адрес
----------	---------------------------	-----	-----	-------	-------

1.	Температура воздуха	Temp_1	аналог	глобал.	%IB0
2.	Кнопка «Пуск»	SB1	дискр.	локал.	-

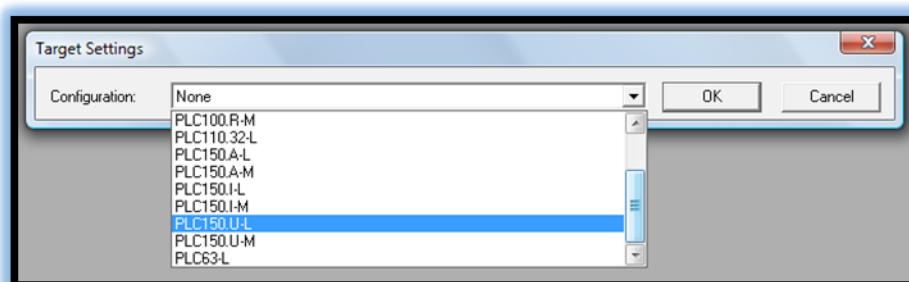
### 3. Создание проекта программы.

При создании проекта используется язык структурированного текста **ST**, реализующий структуры, подобные структурам языка Pascal.

Пользователь запускает **CoDeSys** последовательным выбором приложений:

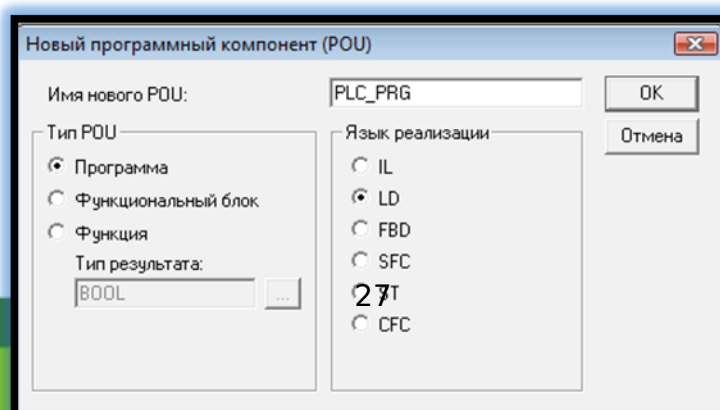
**Пуск → Все программы → 3S Software → CoDeSys V2.3 → CoDeSys V2.3.**

Новый проект открывается из главного меню **File New**. В открывшемся окне (рис. 3.1) выбирается тип контроллера, **PLC150.U-L**, выбор подтверждается нажатием клавиши **OK**.



**Рисунок 3.1-Окно конфигурации «Target Settings» программы**

После выбора проекта выводится экранная форма, задающая тип, имя и язык программирования первичного компонента **New POU**, главной программы контроллера. Необходимо выбрать язык программирования **IL**, установив флаги в позициях, указанных на рис.3.2.





### Рисунок 3.2- Вид окна «New POU» с отмеченными параметрами

*Примечание. Имя главной программы PLC\_PRG и ее тип менять нельзя.*

После подтверждения выбора нажатием клавиши **ОК** откроется окно нового проекта с именем по умолчанию **Untitled**.

#### **4. Параметры входов и выходов.**

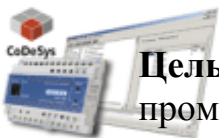
Параметры входов и выходов контроллера задаются на вкладке ресурсов (**Resources**) Организатора объектов **CoDeSys** с помощью утилиты **PLC Configuration (Конфигуратор ПЛК)** (ЛР№1 п.3).

#### **5. Создание программы на языке IL**

При написании программы в рабочей зоне вкладки **POUs** вводится текст, состоящий из имен переменных меток, операторов, операндов, комментариев.

В тексте программы необходимо учесть технологические требования, предъявляемые к системе управления.

## Программирование ПЛК на языке FBD.

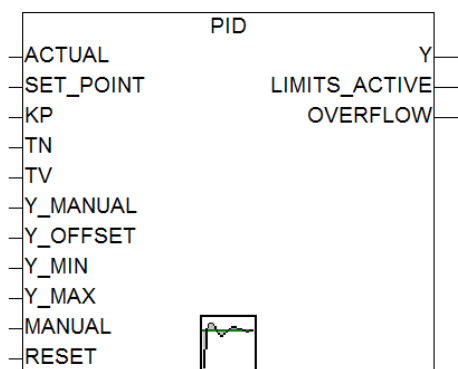


**Цель работы:** Изучить принципы составления прикладных программ для промышленных логических контроллеров на языке FBD пакета CoDeSys.

### 1. Общие сведения

FBD – это графический язык программирования. Он работает с последовательностью цепей, каждая из которых содержит логическое или арифметическое выражение, вызов функционального блока, переход или инструкцию возврата.

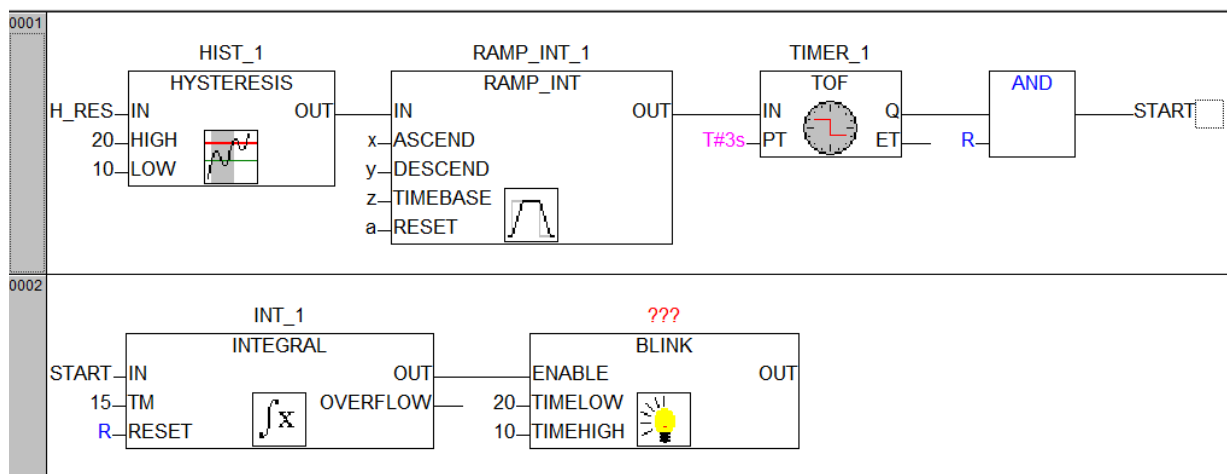
Диаграмма FBD строится из компонентов, отображаемых на схеме прямоугольниками. Входы POU изображаются слева от прямоугольника, выходы справа. Внутри прямоугольника указывается тип POU и наименования входов и выходов. Для экземпляра функционального блока его наименование указывается сверху, над прямоугольником. В графических системах программирования прямоугольник компонента может содержать картинку, отражающую его тип. Размер прямоугольника зависят от числа входов и выходов и устанавливается графическим редактором автоматически.



**Рисунок 4.1- Пример графического представления  
экземпляра функционального блока PID**

Программа в FBD не обязательно должна представлять большую единую схему. Как и в LD, диаграмма образуется из множества цепей, которые выполняются одна за другой.

В CoDeSys все цепи одного POU отображаются в едином графическом окне, пронумерованные и разделенные горизонтальными линиями (рис. 2). Значения переменных, вычисленные в одной цепи, доступны в последующих цепях сразу в том же рабочем цикле.



**Рисунок 4.2 - Диаграмма FBD из двух цепей**

### **Соединительные линии**

Прямоугольники POU в FBD соединены линиями связи. Соединения имеют направленность слева направо. Вход блока может быть соединен с выходом блока, расположенного слева от него. Помимо этого, вход может быть соединен с переменной или константой. Соединение должно связывать переменные или входы и выходы одного типа. В отличие от компонента переменная изображается на диаграмме без прямоугольной рамки. Ширина соединительной линии в FBD роли не играет. Стандарт допускает использование соединительных линий разной ширины и стиля для соединений разного типа.

### **Порядок выполнения FBD**

*Выполнение FBD-цепей идет слева направо, сверху вниз. Блоки,*



расположенные левее, выполняются раньше. Блок начинает вычисляться только после вычисления значений всех его входов. Дальнейшие вычисления не будут продолжены до вычисления значений на всех выходах. Другими словами, значения на всех выходах графического блока появляются одновременно. Вычисление цепи считается законченным только после вычисления значений на выходах всех входящих в нее элементов.

В некоторых системах программирования пользователь имеет возможность свободно передвигать блоки с сохранением связей. В этом случае ориентироваться нужно исходя из порядка соединений. Редактор FBD CoDeSys автоматически расставляет блоки в порядке выполнения.

### **Инверсия логических сигналов**

*Инверсия логического сигнала* в PBD изображается в виде окружности на соединении, перед входом или переменной. Инверсия не является свойством самого блока и может быть легко добавлена или отменена непосредственно в диаграмме.

### **Метки, переходы и возврат**

Порядок выполнения FBD-цепей диаграммы можно принудительно изменять, используя метки и переходы, точно так же, как и в релейных схемах. Метка ставится в начале любой цепи, являясь, по сути, названием данной цепи. Цепь может содержать только одну метку. Имена меток подчинены общим правилам наименования идентификаторов МЭК. Графический редактор автоматически нумерует цепи диаграммы. Эта нумерация применяется исключительно для документирования и не может заменять метки. Переход обязательно связан с логической переменной и

выполняется, если переменная имеет значение **ИСТИНА**. Для создания безусловного перехода используется константа **ИСТИНА**, связанная с переходом.



Оператор возврата **RETURN** можно использовать в FBD так же, как и переход на метку, т. е. в связке с логической переменной. Возврат приводит к немедленному окончанию работы программного компонента и возврату на верхний уровень вложений. Для основной программы это начало рабочего цикла ПЛК.

## 2. Порядок выполнения работы

Из ПРИЛОЖЕНИЯ 2 по заданию преподавателя выбирается вариант технологической схемы.

На основании заданной технологической схемы и описания технологического процесса разработать:

- технологические требования к схеме управления;
- таблицу сигналов;
- прикладную программу для ПЛК.
- дать описание работы прикладной программы.

В таблицу сигналов вносятся:

- порядковый номер переменной;
- имя переменной (не должно содержать пробелов и кириллицы);
- тип переменной (дискретный, аналоговый);
- класс переменной (локальная, глобальная);
- адрес (для внутренних переменных не заполняется).





### Пример заполнения таблицы параметров.

№ п/п	Наименование параметра	Имя	Тип	Класс	Адрес
1.	Температура воздуха	Temp_1	аналог	глобал.	%IB0
2.	Кнопка «Пуск»	SB1	дискр.	локал.	-

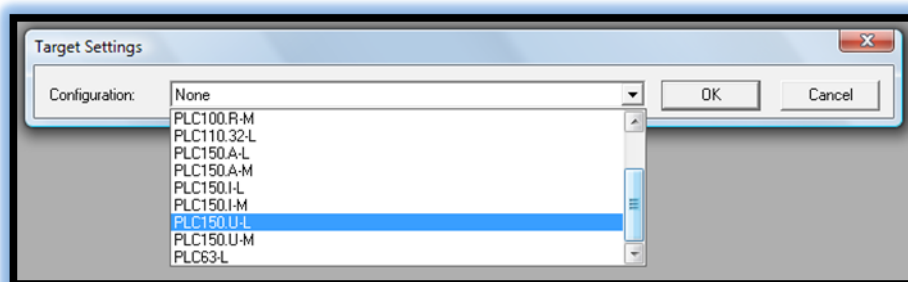
### 3. Создание проекта программы.

При создании проекта используется язык структурированного текста **ST**, реализующий структуры, подобные структурам языка Pascal.

Пользователь запускает **CoDeSys** последовательным выбором приложений:

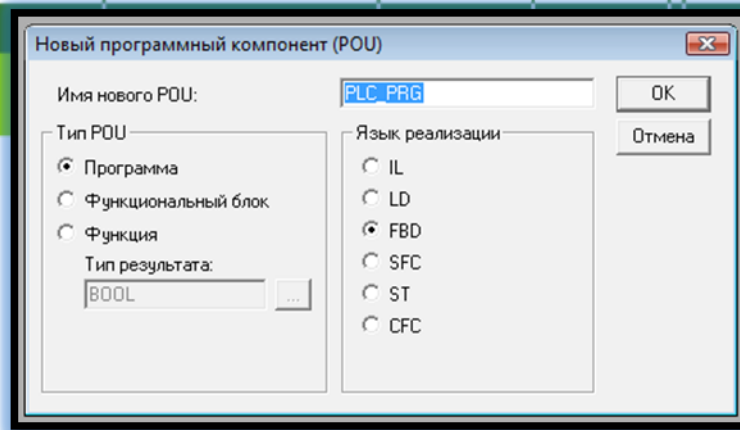
**Пуск → Все программы → 3S Software → CoDeSys V2.3 → CoDeSys V2.3.**

Новый проект открывается из главного меню: **File New**. В открывшемся окне (рис .1) выбирается тип контроллера, **PLC150.U-L**, выбор подтверждается нажатием клавиши **OK**.



**Рисунок 4.3 - Окно конфигурации «Target Settings» программы**

После выбора проекта выводится экранная форма, задающая тип, имя и язык программирования первичного компонента **New POU**, главной программы контроллера. Необходимо выбрать язык программирования **FBD**, установив флаги в позициях, указанных на рис.2.



**Рисунок 4.5 - Вид окна «New POU» с отмеченными параметрами**

***Примечание.** Имя главной программы `PLC_PRG` и ее тип менять нельзя.*

После подтверждения выбора нажатием клавиши **ОК** откроется окно нового проекта с именем по умолчанию **Untitled**.

#### **4. Параметры входов и выходов.**

Параметры входов и выходов контроллера задаются на вкладке ресурсов (**Resources**) Организатора объектов **CoDeSys** с помощью утилиты **PLC Configuration (Конфигуратор ПЛК)** (ЛР№1 п.3).

#### **5. Создание программы на языке FBD**

При написании программы в рабочей зоне вкладки **POUs** вводится текст, состоящий из имен переменных меток, операторов, операндов, комментариев.

В тексте программы необходимо учесть технологические требования, предъявляемые к системе управления.



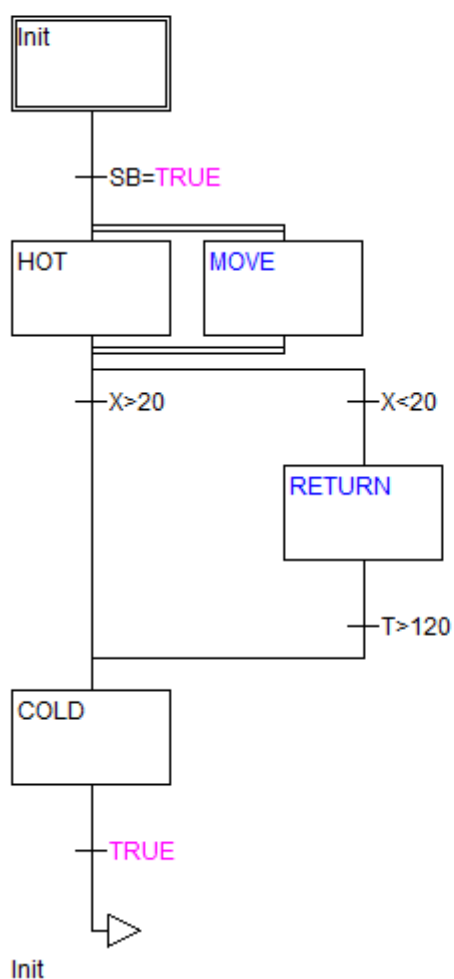
## ЛАБОРАТОРНАЯ РАБОТА 5

### Программирование ПЛК на языке SFC.

**Цель работы:** Изучить принципы составления прикладных программ для промышленных логических контроллеров на языке SFC пакета CoDeSys.

#### 1. Общие сведения

SFC – это графический язык, который позволяет описать хронологическую последовательность различных действий в программе. Для этого действия связываются с шагами (этапами), а последовательность работы определяется условиями переходов между шагами.



## Рисунок 5.1 - Пример SFC диаграммы



Любая SFC-схема составляется из элементов, представляющих шаги и условия переходов (рис.5.1).

Шаги показаны на схеме прямоугольниками. Реальная работа шага (действия) описывается в отдельном окне системы программирования и не отражается на диаграмме. О назначении шага SFC говорит только его название или, если этого не достаточно, краткое текстовое описание (комментарий).

Шаги на схеме могут быть пустыми, что не вызывает ошибки при компиляции проекта. Пустые шаги являются нормой при применении программирования сверху вниз, характерного для SFC. Определить действия, соответствующие шагу, можно в любое время. Нет ничего удивительного, если пустые шаги останутся и в законченном проекте. Задачей пустого шага является ожидание перехода.

### Переходы

Ниже шага на соединительной линии присутствует горизонтальная черта, обозначающая переход. Условием перехода может служить логическая переменная, логическое выражение, константа или прямой адрес.

Переход выполняется при соблюдении двух условий:

- 1) переход разрешен (соответствующий ему шаг активен);
- 2) условие перехода имеет значение TRUE.

Простые условия отображаются непосредственно на диаграмме справа от черты, обозначающей переход. В CoDeSys на диаграмме можно записывать только выражения на языке ST. Для громоздких условий применяется другой подход. Вместо условия на диаграмме записывается только идентификатор перехода. Само же условие описывается в отдельном

окне с применением языка IL, ST, LD или FBD. Переменные или прямые адреса используются в условии перехода только для чтения. В условном



выражении перехода нельзя вызывать экземпляры функциональных блоков и использовать операцию присваивания. Признаком того, что идентификатор перехода на диаграмме является отдельно реализованным условием, а не простой логической переменной, служит закрашенный угол перехода. В качестве условия перехода может быть задана логическая константа. Если задано TRUE, то шаг будет выполнен однократно, за один рабочий цикл, далее управление перейдет к следующему шагу. Если задано условие FALSE, то шаг будет выполняться бесконечно.

### **Начальный шаг**

Каждая SFC-схема начинается с шага, выделенного графически двойными вертикальными линиями или по всему периметру. Это - *начальный шаг*. Наименование начального шага может быть произвольным (по умолчанию Init). Начальный шаг присутствует обязательно, хотя и может быть пустым.

### **Параллельные ветви**

Несколько ветвей SFC могут быть параллельными. Признаком параллельных ветвей на схеме является двойная горизонтальная линия. Каждая параллельная ветвь начинается и заканчивается шагом. То есть условие входа в параллельность всегда одно, условие выхода тоже одно на всех.

Параллельные ветви выполняются теоретически одновременно. В жизни это означает - в одном рабочем цикле, слева направо. Условие перехода, завершающее параллельность, проверяется только в случае, если в каждой параллельной ветви активны последние шаги.

### **Альтернативные ветви**

Несколько ветвей SFC могут быть альтернативными ветвями.

Признаком альтернативных ветвей на схеме является одинарная горизонтальная линия. Каждая альтернативная ветвь начинается и



заканчивается собственным условием перехода. Проверка альтернативных условий выполняется слева направо. Если верное условие найдено, то прочие альтернативы не рассматриваются. В альтернативных ветвях всегда работает только одна из ветвей, поэтому ее окончание и будет означать переход к следующему за альтернативной группой шагу. При создании альтернативных ветвей желательно задавать взаимоисключающие условия. В этом случае вероятность допустить ошибку при анализе или в процессе доработки диаграммы значительно ниже.

### **Переход на произвольный шаг**

В общем случае SFC-схема выполняется сверху вниз. Стандартом допускается создание переходов на произвольный шаг. Для этого применяются соединительные линии с промежуточными стрелками или поименованные переходы. То есть переход выполняется на шаг, имя которого указано под стрелкой. В англоязычных источниках переход на произвольный шаг называется «прыжок» (jump).

Прыжок из одной ветви параллельного блока наружу вызывает эффект размножения маркера. Прыжок внутрь параллельного блока нарушает параллельность ветвей. Подобных трюков необходимо избегать.

## **2. Порядок выполнения работы**

Из ПРИЛОЖЕНИЯ 2 по заданию преподавателя выбирается вариант технологической схемы.

На основании заданной технологической схемы и описания технологического процесса разработать:

- технологические требования к схеме управления;
- таблицу сигналов;

- прикладную программу для ПЛК;
- дать описание работы прикладной программы.



В таблицу сигналов вносятся:

- порядковый номер переменной;
- имя переменной (не должно содержать пробелов и кириллицы);
- тип переменной (дискретный, аналоговый);
- класс переменной (локальная, глобальная);
- адрес (для внутренних переменных не заполняется).

#### Пример заполнения таблицы параметров.

№ п/п	Наименование параметра	Имя	Тип	Класс	Адрес
1.	Температура воздуха	Temp_1	аналог	глобал.	%IB0
2.	Кнопка «Пуск»	SB1	дискр.	локал.	-

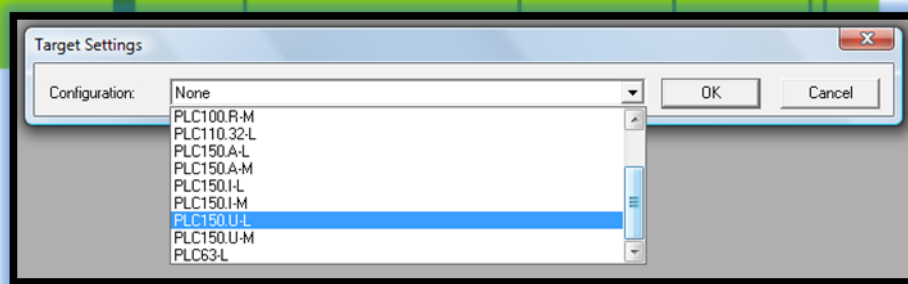
### 3. Создание проекта программы.

При создании проекта используется язык структурированного текста **ST**, реализующий структуры, подобные структурам языка Pascal.

Пользователь запускает **CoDeSys** последовательным выбором приложений:

**Пуск → Все программы → 3S Software → CoDeSys V2.3 → CoDeSys V2.3.**

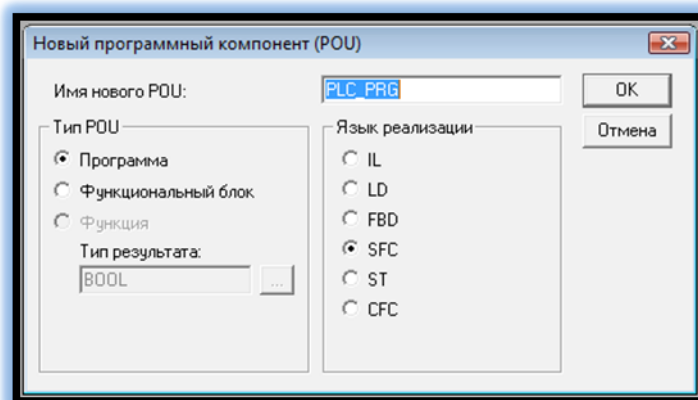
Новый проект открывается из главного меню: **File New**. В открывшемся окне (рис.5 .1) выбирается тип контроллера, **PLC150.U-L**, выбор подтверждается нажатием клавиши **ОК**.



**Рисунок 5.1 - Окно конфигурации «Target Settings» программы**

После выбора проекта выводится экранная форма, задающая тип, имя и язык программирования первичного компонента **New POU**, главной

программы контроллера. Необходимо выбрать язык программирования SFC, установив флаги в позициях, указанных на рис.5.2.



**Рисунок 5.2 - Вид окна «New POU» с отмеченными параметрами**

***Примечание.** Имя главной программы PLC\_PRG и ее тип менять нельзя.*

После подтверждения выбора нажатием клавиши **ОК** откроется окно нового проекта с именем по умолчанию **Untitled**.

#### **4. Параметры входов и выходов.**

Параметры входов и выходов контроллера задаются на вкладке ресурсов (**Resources**) Организатора объектов **CoDeSys** с помощью утилиты **PLC Configuration (Конфигуратор ПЛК)** (ЛР№1 п.3).

#### **5. Создание программы на языке SFC**



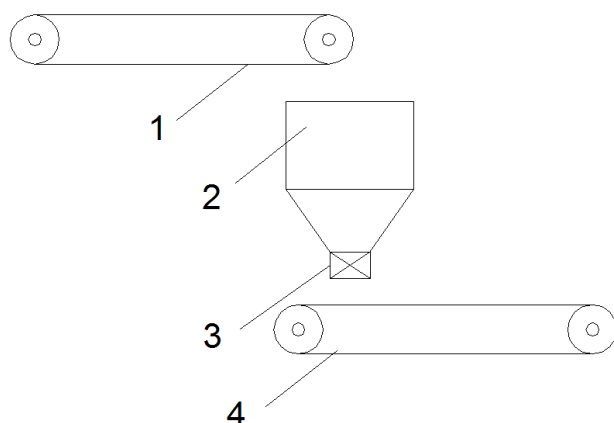
При написании программы в рабочей зоне вкладки **POUs** вводится

текст, состоящий из имен переменных меток, операторов, операндов, комментариев.

В тексте программы необходимо учесть технологические требования, предъявляемые к системе управления.

## ПРИЛОЖЕНИЕ 1

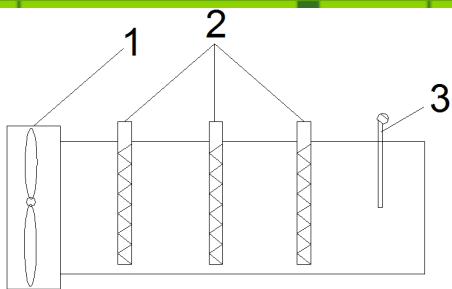
### Задание 1.



#### *Линия дозации продукта*

Продукт с помощью загрузочного транспортера 1 попадает в бункер 2. Транспортер работает до тех пор, пока вес продукта в бункере не станет больше заданного. Затем транспортер 1 останавливается, срабатывает задвижка 3 и включается транспортер 4. После разгрузки бункера, задвижка закрывается, транспортер 4 останавливается и загрузка начинается вновь.

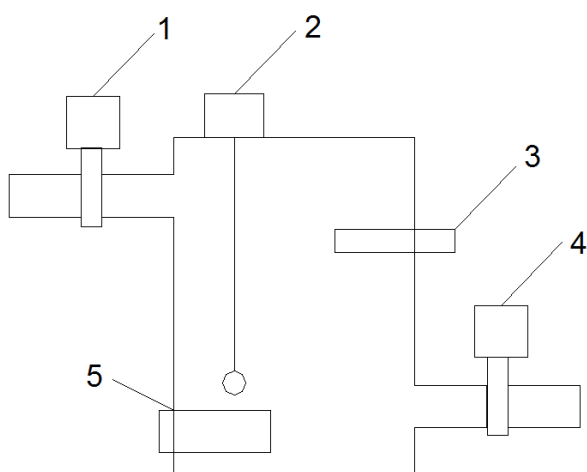
### Задание 2.



### ***Тепловая пушка***

Воздух вентилятором 1 прогоняется через тепловую пушку. В зависимости от установки температуры включается определенное количество нагревательных элементов 2. Следует учесть, что нагревательные элементы не должны работать при выключенном вентиляторе. 3-измеритель температуры.

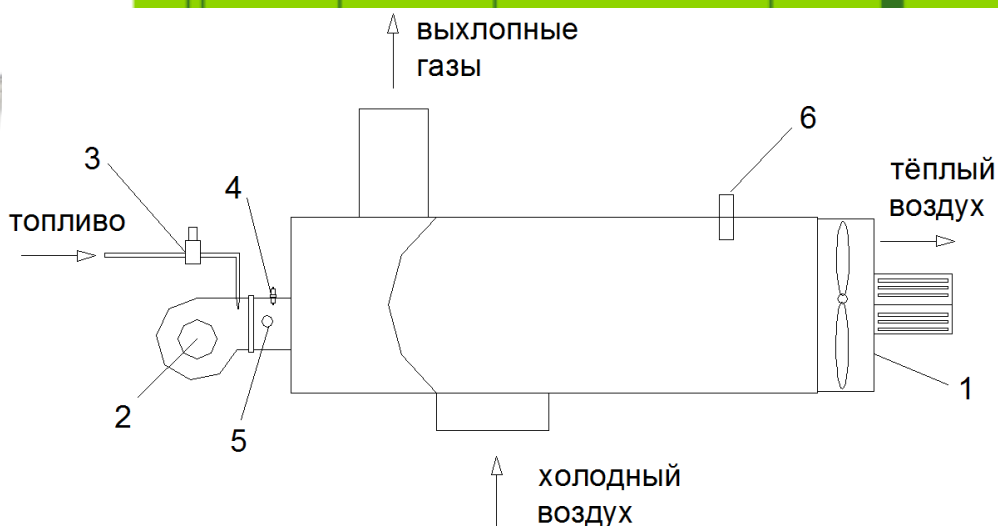
### **Задание 3**



### ***Водонагревательная установка***

Вода через заливной клапан 1 заполняет ёмкость до определенного уровня, измеряемого датчиком уровня 2. Вода ТЭНом 5 нагревается до заданной температуры, измеряемой датчиком температуры 3, и сливается через сливной клапан 4.

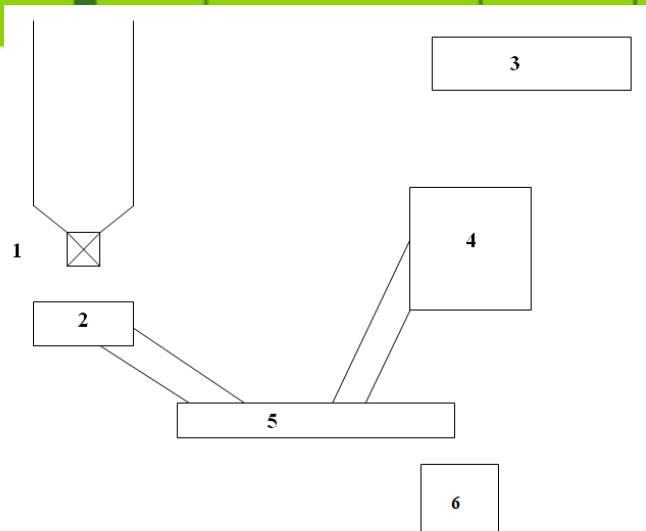
### **Задание 4.**



### ***Теплогенератор***

При нажатии на кнопку пуск, звучит предупредительная сигнализация и запускается основной вентилятор теплого воздуха 1. После запуска основного вентилятора, включается топливный вентилятор 2 для продувки (10 с). Затем включается топливный соленоидный клапан 3 и топливная смесь закачивается в камеру сгорания (5 с). Срабатывает запальная свеча 4 (4 с). Реле пламени 5 контролирует наличие пламени. Если пламя не появилось в течение 5 с., процесс розжига выполняется еще раз (с продувки воздухом 15с.). При повторном незапуске агрегата включается продувка 1 мин. и аварийная сигнализация. При нормальном запуске агрегата, система должна контролировать температуру воздуха на выходе термопреобразователем 6 и изменять скорость вращения топливного вентилятора 2. При остановке агрегата, продувка должна осуществляться до тех пор, пока температура не упадет ниже  $T_{min}$ .

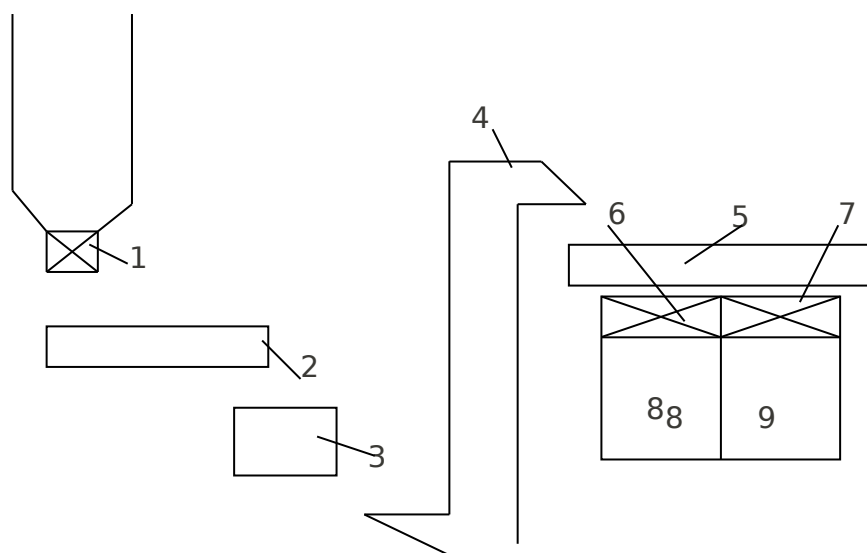
#### **Задание 5.**



## Описание технологического процесса

Зерно через задвижку 1 поступает на дробилку 2 и далее на транспортер-смеситель 5. Сюда же поступают переработанные в мойке-корнерезке 4 корнеплоды (3 транспортер нарезанных корнеплодов). Транспортером смесителем 5 смесь загружается в смеситель 6. Предусмотреть совместную и раздельную работу линий зерна и корнеплодов.

### Задание 6.

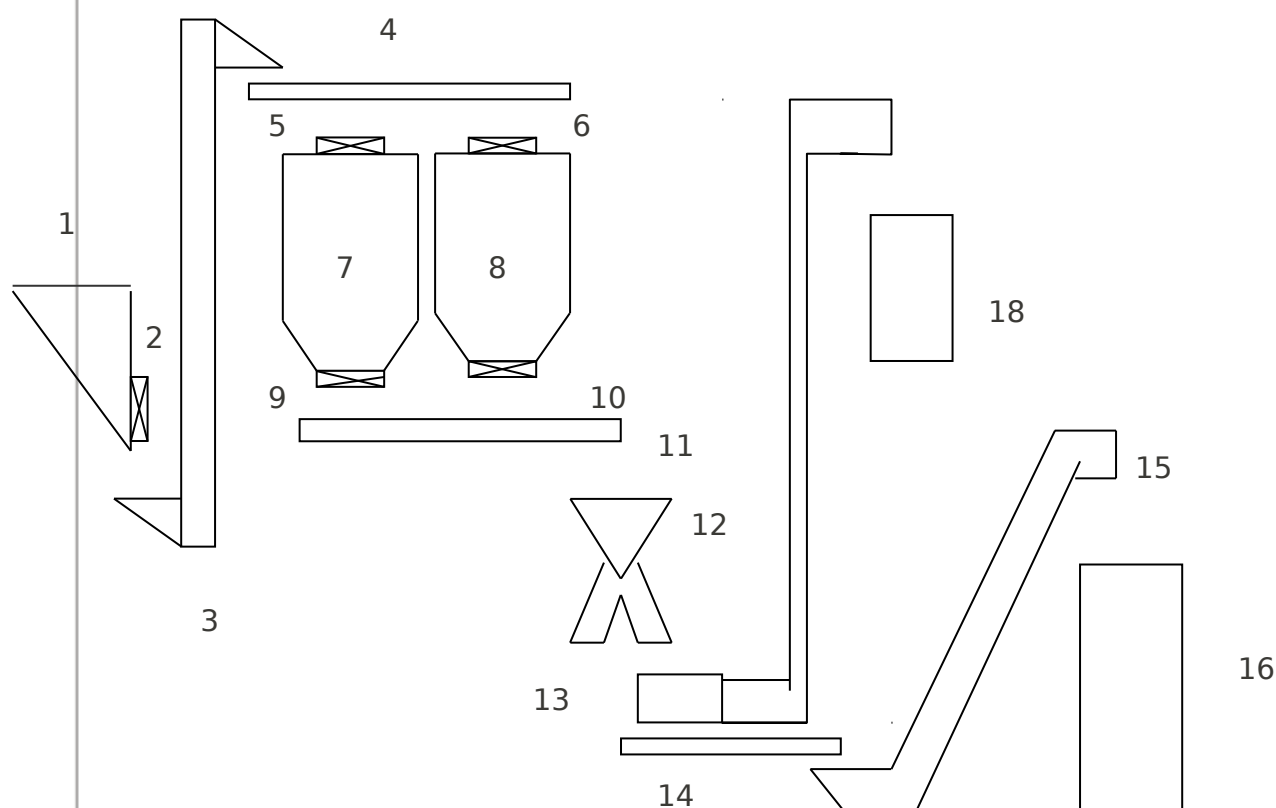


## Описание технологического процесса

Зерно из бункера через задвижку 1 поступает на транспортер 2 и далее в дробилку 3. Измельченное зерно норией 4 подается на шнековый

транспортёр 5 и далее либо в бункер 8 либо в бункер 9. Линия должна отключиться при заполнении одного из бункеров. Режим работы электродвигателей поточной линии кратковременный.

### Задание 7.



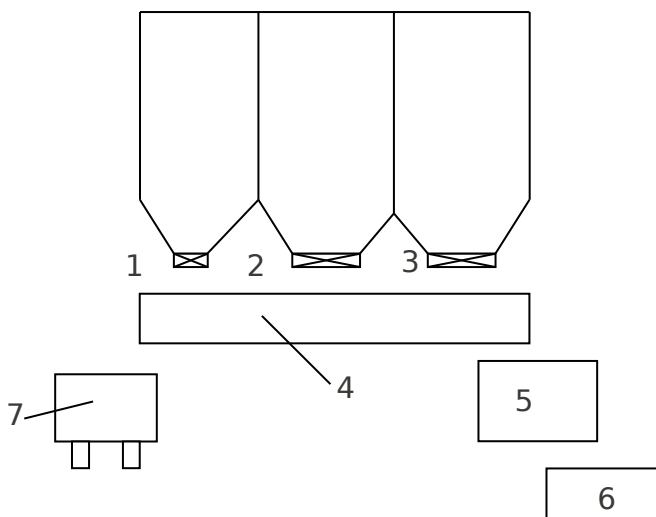
### Описание технологического процесса

Из завальной ямы 1 семечки через задвижку 2 норией подаются на шнековый транспортер и затем через задвижку 5 и 6 заполняют бункера 7 и 8. Из бункеров 7 и 8 через задвижки 9 и 10 семечки поступают на наклонный транспортер 11, который заполняет жим 12.

После жима масло из накопительной емкости насосом 13 подается в емкость 18. Жмых после отжима поступает на транспортер 14 и далее норией 15 загружается в накопительный бункер 16. Режим работы двигателей кратковременный.



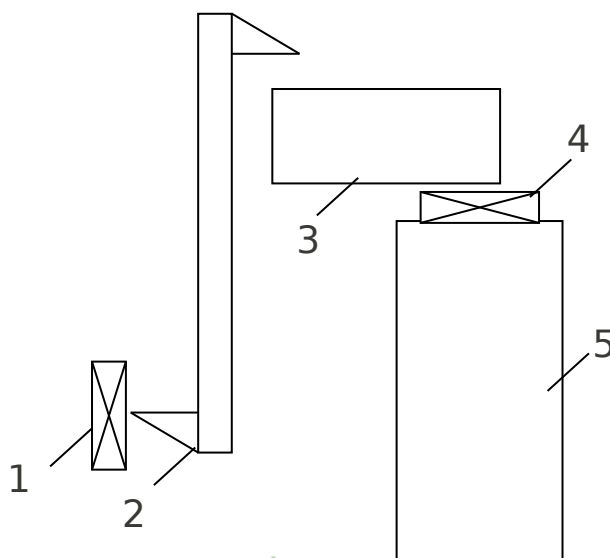
### Задание 8.



### Описание технологического процесса

Зерно поступает на транспортер 4 через одну из задвижек 1,2 или 3 или все вместе (выбор задвижки производится оператором) и далее либо в тележку 7 либо на дробилку 5 и далее в бункер 6. Схема должна отключаться при срабатывании датчика уровня в бункере 6 или при срабатывании датчика давления под тележкой.

### Задание 9.

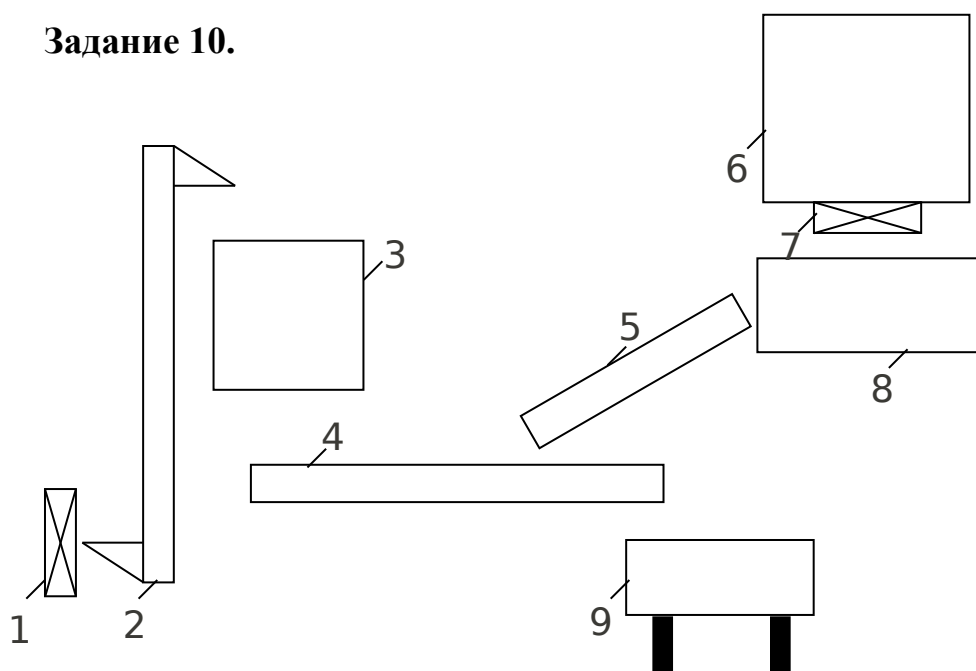




### Описание технологического процесса

Зерно из завальной ямы через заслонку 1 норией 2 подается на дробилку 3, где оно измельчается. Измельченное зерно через заслонку 4 загружается в бункер 5. Предусмотреть отключение схемы в рабочем порядке и при срабатывании датчиков уровня. Двигатели технологической схемы работают в кратковременном режиме.

#### Задание 10.



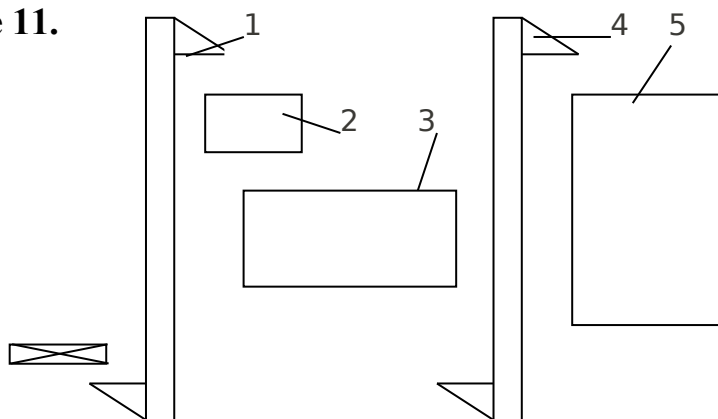
#### Описание технологического процесса

Технологическая линия состоит из линии переработки зерна и линии переработки корнеплодов. В состав линии переработки зерна входят задвижка 1 в завальной яме, нория 2, дробилка 3. Линия переработки корнеплодов содержит бункер нерезанных корнеплодов 6, задвижку бункера 7, мойку корнеплодов 8, транспортер измельченных корнеплодов 9. Продукты с обеих линий поступают на транспортер смеситель 4 и далее

загружаются в тележку 9. Предусмотреть раздельную и совместную работу линий переработки зерна и корнеплодов.



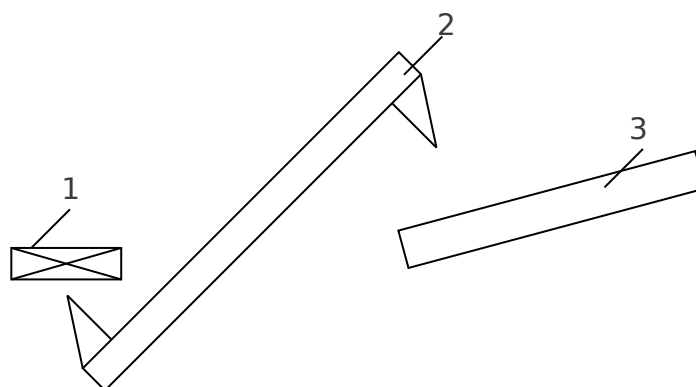
### Задание 11.



### Описание технологического процесса

Зерно из завальной ямы норией 1 подается на триерный блок 3. Очищенное зерно норией 4 загружается в бункер 5. Предусмотреть работу линии с очисткой зерна и без очистки.

### Задание 12.



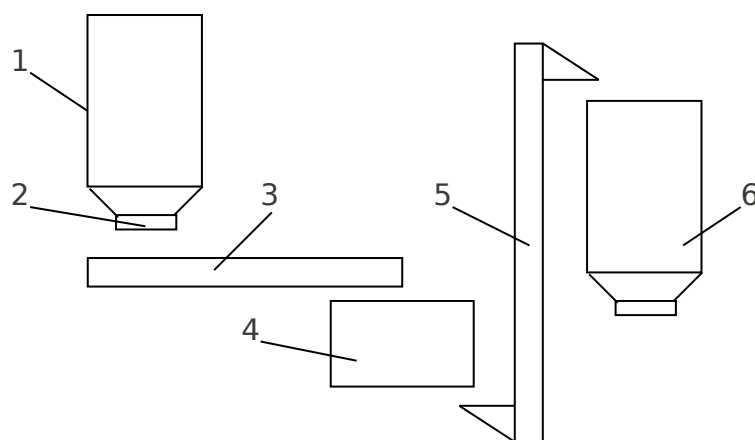
### Описание технологического процесса



Зерно через заслонку 1 норией 2 подается на метательный транспортер 3.



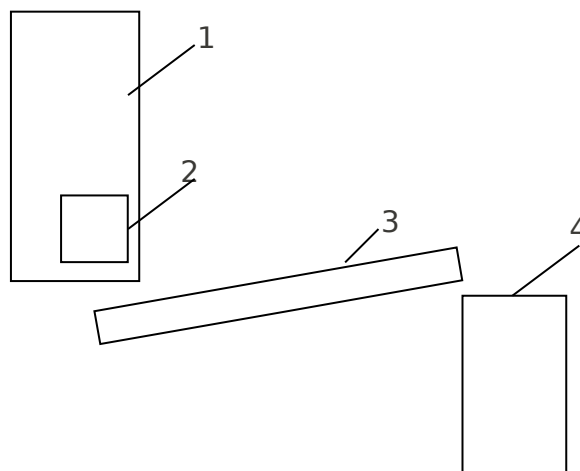
### Задание 13.



### Описание технологического процесса

Зерно из бункера 1 через заслонку 2 шнековым транспортером 3 подается на мельницу 4. Продукт помола норией 5 подается в бункер 6. Предусмотреть отключение линии при заполнении бункера по сигналу датчика уровня.

### Задание 14.

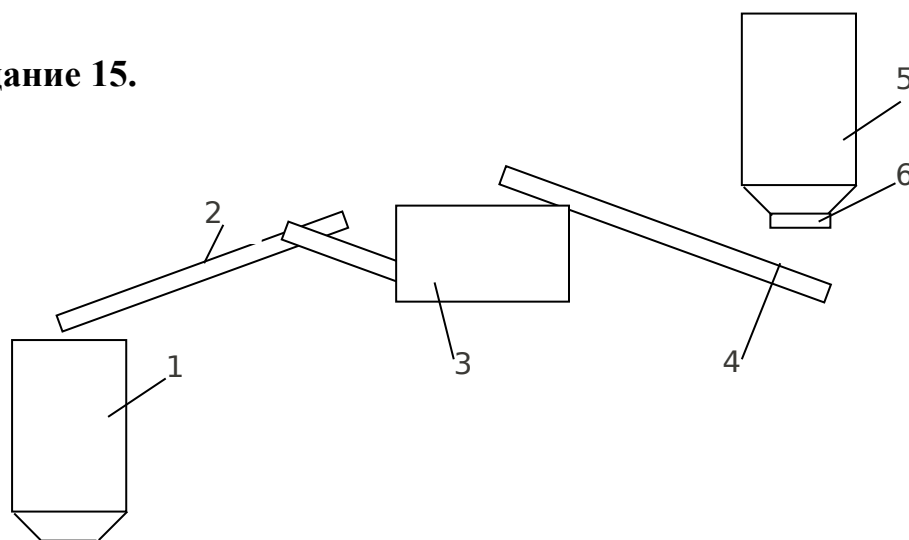


### Описание технологического процесса

Продукт из бункера 1 шнековым дозатором корма 3 подается в бункер дозатор кормораздатчика 4. Предусмотреть отключение линии при срабатывании датчика уровня в бункере дозаторе 4. Для исключения образования сводов при хранении корма предусматривается вибратор 2.



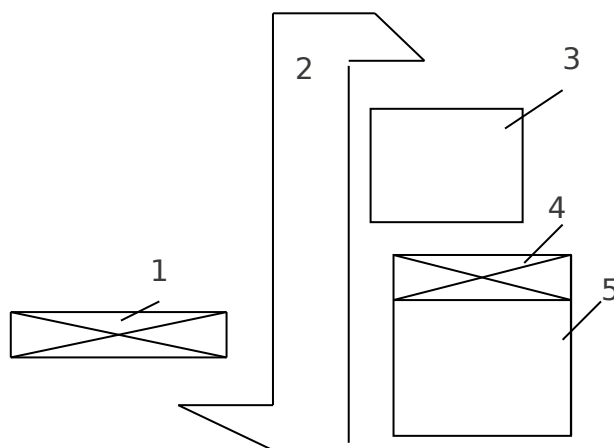
### Задание 15.



### Описание технологического процесса

Корнеплоды из бункера 5 через электромагнитную заслонку 6 поступают на скребковый транспортер 4 ТК-5Б, который производит загрузку корнеклубней 3. Измельченные корнеплоды шнековым транспортером 2 ШЗС-40 загружаются в смеситель 1.

### Задание 16.

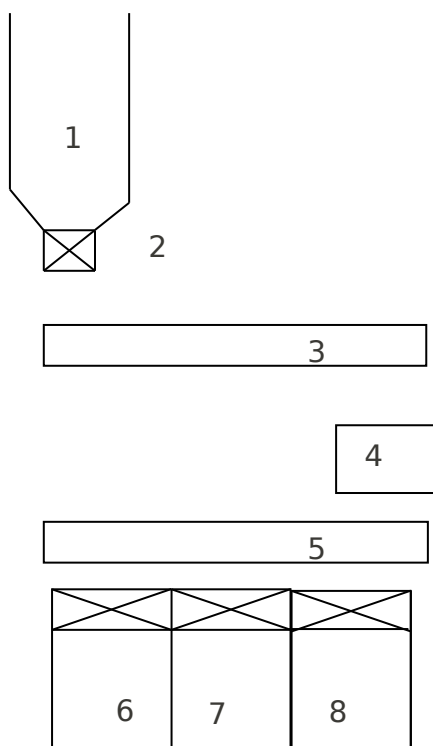


### Описание технологического процесса

При открытии заслонки 1 продукт норией 2 подается в дробилку 3. Измельченный продукт из дробилки через заслонку 4 заполняет бункер 5. Предусмотреть отключение линии при заполнении бункера по сигналу датчика уровня.



### Задание 17.

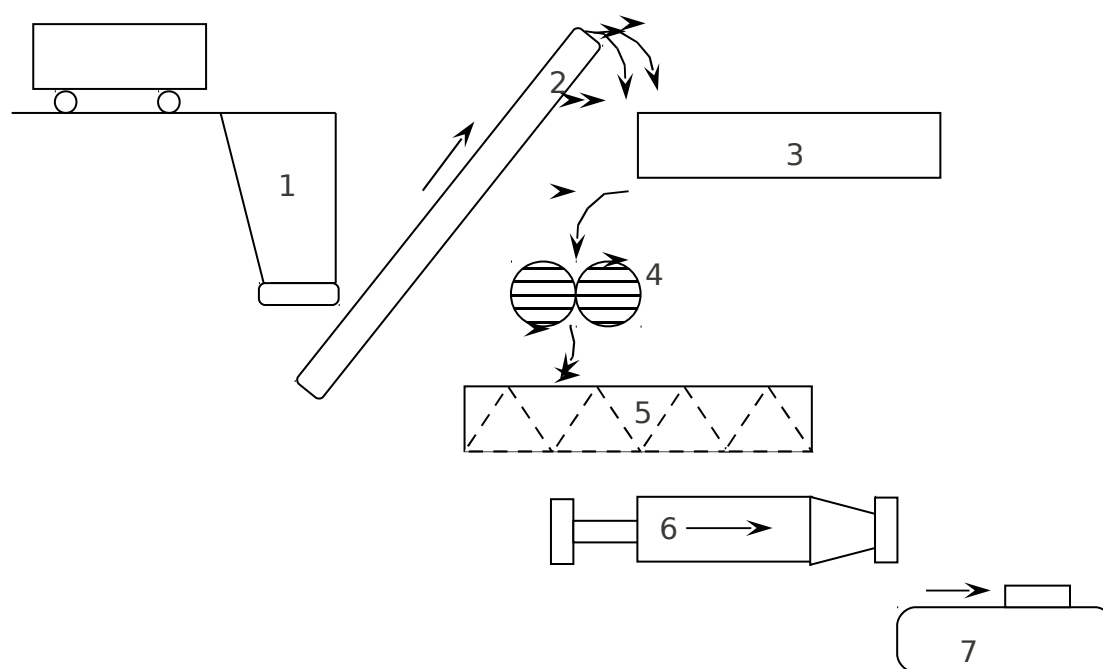


### Описание технологического процесса

Зерно из бункера 1 через заслонку 2 шнековым транспортом 3 подается на дробилку 4. Измельченный продукт транспортом 5 через электромагнитные заслонки 6, 7, 8 загружается в один из бункеров. Выбор бункера осуществляется оператором.

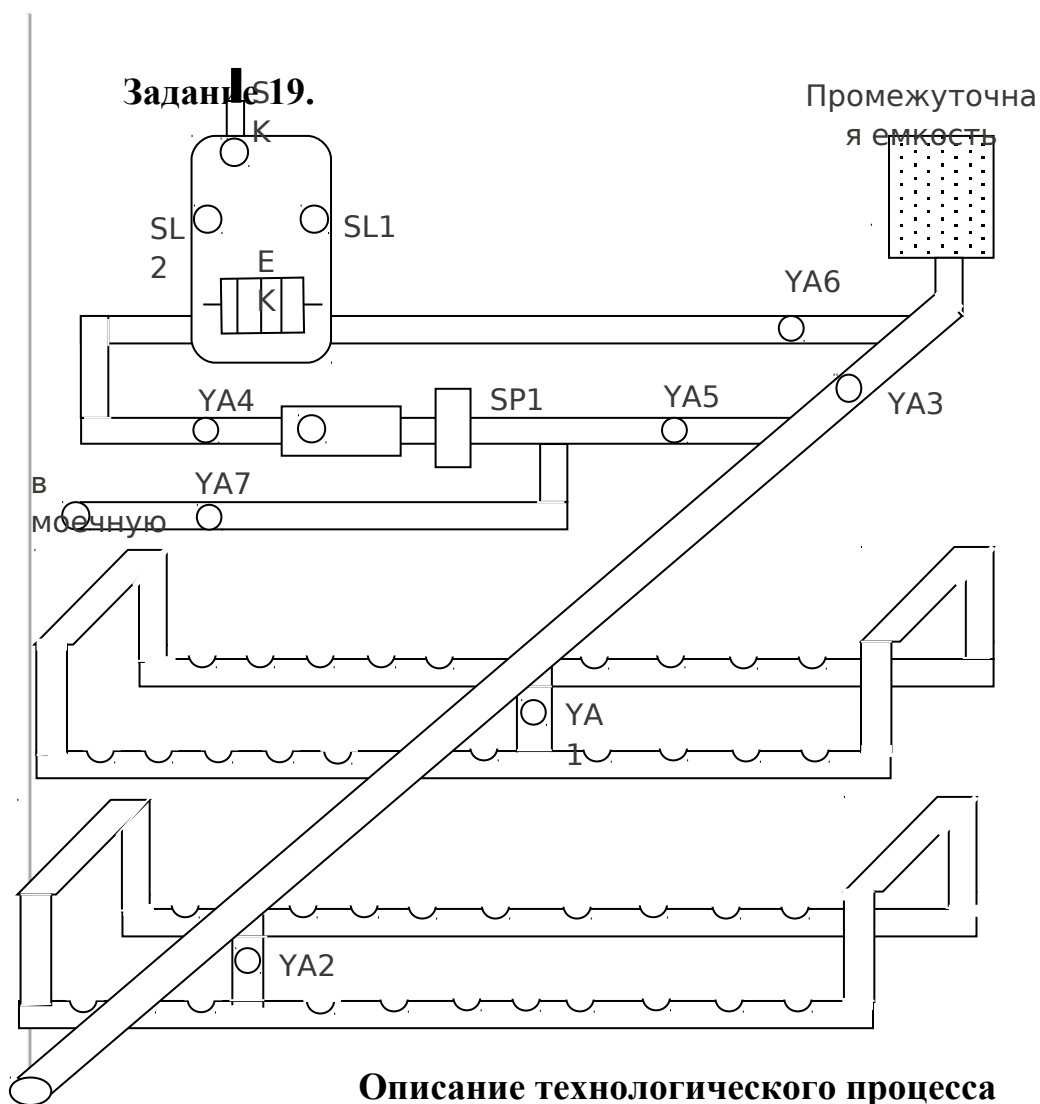


### Задание 18.



### Описание технологического процесса

Глина из завальной ямы 1 транспортером подается на камневывделительные валцы 3. Далее глина будет проходить через гладкие валцы 4 и поступать в глиномешалку 5. Глина прессом 6 выдавливается и поступает на резательный механизм 7.



### Описание технологического процесса

SP1

Система поения воды должна предусматривать следующие режимы работы: подача воды в систему поения без подогрева в летний период; подача в систему поения подогретой воды в зимний период; подача подогретой воды в моечную.

В летний период вода поступает в систему поения через электромагнитные клапаны YA1, YA2, YA3.

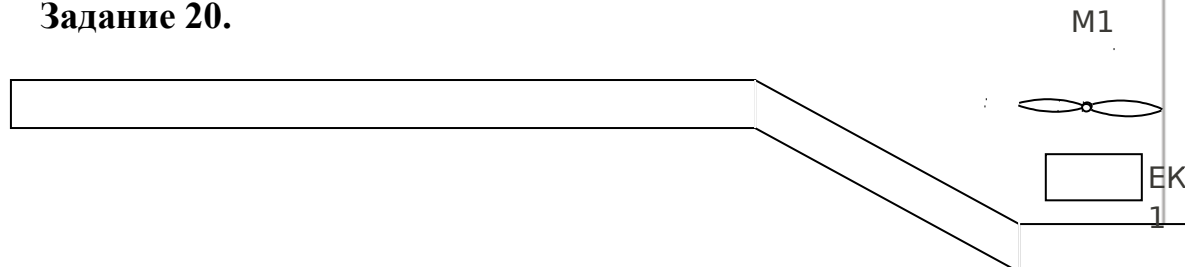
В зимний период вода через заслонку YA6 при закрытой заслонке YA3 вода поступает в водонагреватель EK1. Водонагреватель включается при его заполнении. Контроль за уровнем воды в водонагревателе осуществляется манометрическими датчиками уровня. Когда вода достигает



заданной температуры, водонагреватель отключается, включается насос и подает воду в систему поения через открытую заслонку YA5. Контроль за давлением воды в системе осуществляется с помощью датчиков давления SP1, SP2.

Аналогичным образом система работает в том случае, если подогретую воду необходимо подавать в моечную. Отличие состоит в том, что вода в моечную поступает через заслонку YA7 при закрытой заслонке YA5.

#### **Задание 20.**

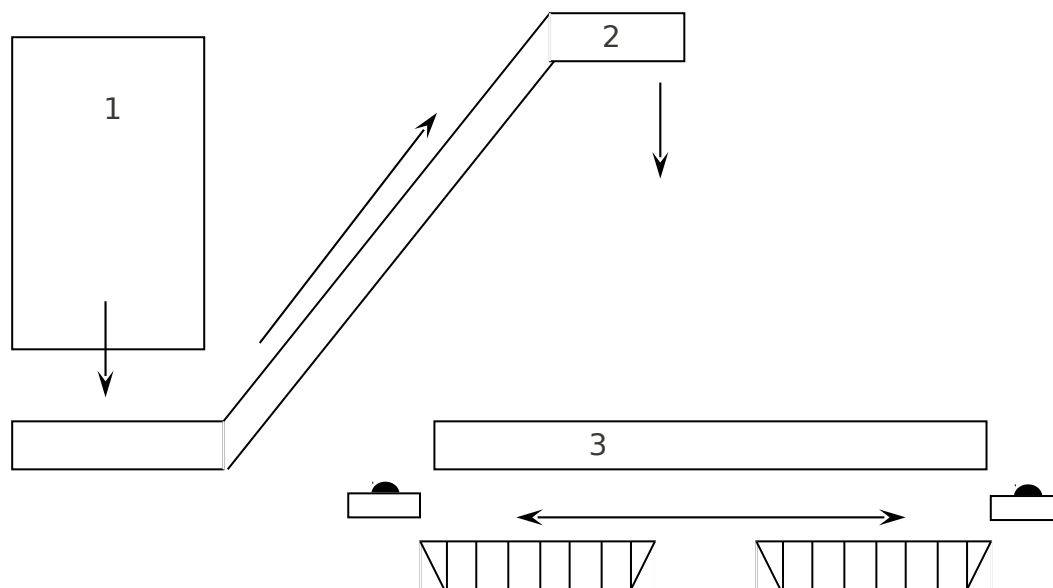


### **Описание технологического процесса**

Разработать схему управления отопительно-вентиляционной установкой.

Подача воздуха в отопительно-вентиляционную систему осуществляется вентилятором. В холодное время года воздух подогревается калорифером. Теплый воздух в помещение попадает через систему воздуховодов. В том случае, если температура воздуха в помещении понижается двигатель вентилятора переходит на пониженную частоту вращения.

### Задание 21.



### Описание технологического процесса

Продукт на платформенный раздатчик корма 3 подается загрузочным транспортером 2 и шнековым дозатором корма из бункера 1. Платформенный раздатчик начинает движение после того, как на него падает первая порция корма. При этом транспортер 3 движется вправо. При наезде на конечный выключатель SQ1 корм сбрасывается в кормушки и транспортер останавливается. Обратное движение платформенного раздатчика начинается через одну-две секунды, при этом происходит заполнение второй половины платформенного раздатчика. Через выдержку времени должно произойти отключение шнекового дозатора корма, а остатков корма на загрузочном транспортере 2 должно хватить для заполнения оставшейся части фронта кормления. При наезде на конечный выключатель SQ2 происходит сбрасывание корма во вторую половину

кормушек и отключение всей схемы. Сброс корма в кормушки производится плужковыми сбрасывателями.



## СОДЕРЖАНИЕ

1. Лабораторная работа №1.	
<i>Программирование ПЛК на языке LD</i>	1
2. Лабораторная работа №2.	
<i>Программирование ПЛК на языке ST</i>	10
3. Лабораторная работа №3.	
<i>Программирование ПЛК на языке IL</i>	20
4. Лабораторная работа №4.	
<i>Программирование ПЛК на языке FBD</i>	28
5. Лабораторная работа №5.	
<i>Программирование ПЛК на языке SFC</i>	34
Приложение №1	40





## ЛИТЕРАТУРА

1. Петров И.В., Дьяконов В.П. Программируемые контроллеры. Стандартные языки и приемы прикладного проектирования./ И.В.Петров, В.П.Дьяконов.– М.: СОЛОН-Пресс2004.
2. И.Г.Минаев, В.В.Самойленко Программируемые логические контроллеры. – Ставрополь: Изд-во «АГРУС», 2009.
3. Руководство пользователя по программированию ПЛК в CoDeSys 2.3. 3S - Smart Software Solutions GmbH, ПК Пролог.-2006.

