Министерство образования Российской Федерации

Уральский государственный технический университет

**Кафедра Автоматика и управление в технических системах**

Зав. каф. В.Г. Лисиенко

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка проекта

### **Микропроцессорная система управления, предназначенная для использования на лесопильном заводе**

**Курсовой проект**

Пояснительная записка

1909 420 000 030 ПЗ

Руководитель Мокрецов В.П.

Студент Данилов Д.

Р-311а Молоков Т.

Екатеринбург 2006

**Содержание**

Постановка задачи

Описание принципа действия системы

Аппаратная часть микропроцессорной системы

Программная часть микропроцессорной системы

Заключение

**Постановка задачи**

Требуется разработать микропроцессорную систему управления, предназначенную для использования на лесопильном заводе. Лесопильная установка имеет конвейер, который доставляет бревна к распилочным механизмам. Бревна лежат на ленте конвейера вдоль нее по одному на ширину ленты.

Искомая микропроцессорная система (в дальнейшем просто МП система) должна нести следующие функции и иметь свойства:

* Измерить диаметр каждого бревна, проходящего по конвейеру, и, если он не укладывается в рамки [20…50]см, то считать бревно бракованным и не подавать его на распилку;
* Измерить длину *lбр* каждого бревна и вычислить его объем *Vi*; суммируя объемы неотбракованных бревен, подсчитывать общий объем распиленной древесины *VΣ*;
* По включении питания МП система должна проводить самотестирование (POST – power on self test) и самоинициализацию;
* Система должна включать ОЗУ, ПЗУ, часы реального времени;
* Информация должна отображаться на восьми буквенно-цифровых индикаторах.

Цель курсовой работы – спроектировать аппаратную и программную части МП системы. Аппаратную основу системы составляет микропроцессор Z80 фирмы Zilog. В задачу курсовой работы не входит подробное проектирование приводов и других механизмов, однако их рассмотрение потребуется для написания управляющей программы.

**Описание принципа действия системы**

###### **Общая структура**

На рис. 1 изображена схема “вход-выход” МП системы.

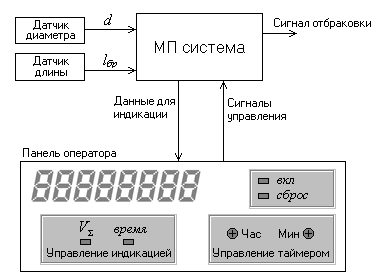


Рис. 1 Структура “вход-выход” МП системы

Цифровые сигналы с датчиков диаметра и длины бревна поступают в систему, формируется сигнал отбраковки. Затем система, имитируя калькулятор, рассчитывает *Vi* и обновляет сумму *VΣ*, храня эти величины в памяти. В зависимости от двух кнопок управления индикацией (одна из них, будучи нажатой, переводит в отжатое положение другую) на индикаторах отображается *VΣ* или текущее время. Для начальной установки часов есть 2 кнопки управления таймером, увеличивающие на единицу часы и минуты. Имеется кнопка включения/выключения питания “Вкл”.

**Обзор датчиков и способа отбраковки**

# Чтобы вычислить объем бревна *Vi*, МП система должна “знать” формулу для вычисления. Формула для расчета *Vi* (в предположении, что бревно – круглый цилиндр):

. (1)

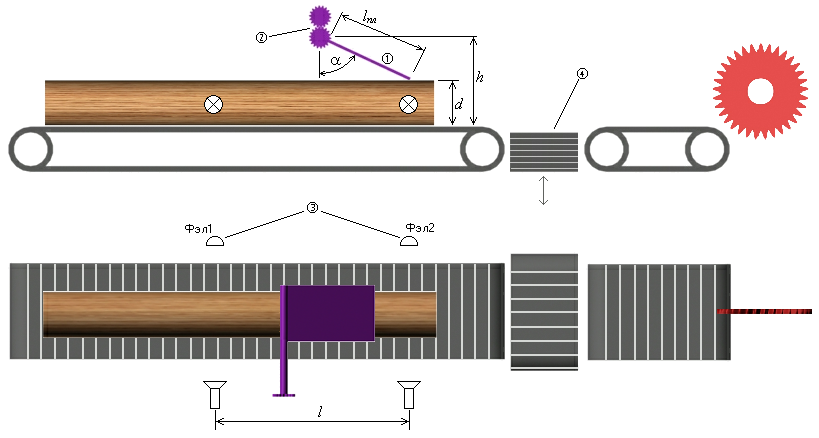
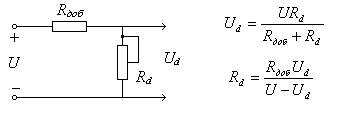


Рис. 2 Конструкция датчиков и механизма отбраковки

Чтобы найти диаметр *d* и длину *lбр*, надо знать конструкцию датчиков (рис. 2).

Датчик диаметра устроен следующим образом: бревно, двигаясь по конвейеру, отклоняет пластину ➀, вращающуюся на горизонтальной оси, на угол α. Этот угол через редуктор ➁ подается на ось переменного резистора. Переменный резистор *Rd* включен в потенциометрический датчик (рис.3). Напряжение *Ud* с этого датчика подается в МП систему, где оцифровывается с помощью АЦП.



## Рис. 3 Потенциометрический датчик

### Расчет диаметра идет по формуле:

### *d = h – lпл* cos α, т.е. надо знать угол α. (2)

### Микропроцессор же имеет лишь напряжение *Ud*. По нему надо рассчитать текущее сопротивление переменного резистора *Rd* (см. формулу на рис.3), из него найти угол поворота оси резистора, которому соответствует это текущее сопротивление, из него, зная передаточное отношение редуктора, определить α.

### Примем следующие допущения:

### номинал *Rдоб*=1 КОм; номинал *Rd* (переменное)=5,1 КОм; напряжение питания датчика *U*=12 В. Примечание: при этом диапазон *Ud* составляет [0; 10]В.

### особенности конструкции – а именно крепления переменного резистора и передаточного отношения редуктора – таковы, что

### если α=0 (пластина вертикальна), то текущее *Rd* =0;

### если α=90° (пластина горизонтальна), то *Rd* =*Rd*max=5,1 КОм (максимально).

Примечание: данное требование однозначно определяет передаточное отношение редуктора – оно будет равно *i*1-2=90°/*α*max, где *α*max – угловой диапазон поворота оси резистора в градусах (т.к. при повороте пластины на 90° ось резистора поворачивается на *α*max)

Это допущение означает, что угол α может быть вычислен из максимального значения переменного резистора как

.

* последнее допущение: пусть *h*=62,5см=0,625м; *lпл*=50см=0,5м.

### Получается формула перехода от *Ud* к α:

 (3)



### Напряжение с датчика *Ud* в системе измеряется с помощью 10-разрядного АЦП, рассчитанного на максимальное входное напряжение 10,5 В. Микропроцессор читает с АЦП не напряжение в вольтах, а его код – число в диапазоне [0..3FFH]=[0..102310]. Поэтому формулу (3) надо изменить, преобразовав напряжение 12 вольт в такой же код, исходя из интервала дискретности АЦП. Он равен д=10,5/1023=1,026⋅10-2 В. Отсюда 12 вольт будет изображаться величиной 12/ д ≈1169 единиц.

### Итоговая формула перехода от *Ud* к α:

### [рад] (4)

### Подставим числа в формулу для диаметра (2):

*d =* 0,625 – 0,5cos α (5)

### Сделаем вывод. Микропроцессор для измерения диаметра делает такие шаги. Зная *Ud* в виде кода, находит угол α в радианах – формула (4). Зная α, находит диаметр *d* – формула (5).

Датчик длины бревна основан на двух фотоэлементах Фэл1 и Фэл2 (➂ на рис.2), поставленных вдоль конвейера на расстоянии *l*=1м друг от друга и двух таймерах внутри микропроцессорной системы. Фотоэлементы регистрируют смену освещенности “свет–темнота” и “темнота–свет”. Принципиально замер длины основан на предположении, что скорость ленты конвейера *v* постоянна.

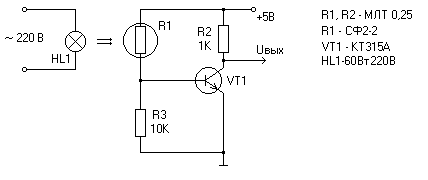
#### Таймеры замеряют два интервала времени:

T1 – период времени, в течение которого передний край бревна проползет от Фэл1 к Фэл2;

Т2 – период времени, в течение которого бревно полностью проползет Фэл1.

 =>  (6)

Фотоэлемент устроен следующим образом (рис. 4):



##### Рис.4 Принципиальная схема фотоэлемента

Лампа HL1 – обычная лампа накаливания, ее свет сфокусирован отражателем на “своем” светоприемнике. Если свет доходит до светоприемника (бревно не загораживает) то транзистор открыт, на выходе фотоэлемента низкий уровень (0), иначе (в темноте) на выходе высокий уровень (1). Выходы Фэл1 и Фэл2 являются входами МП системы.

Алгоритм измерения длины бревна будет таким. Последовательно:

если на Фэл1 перепад 0-1 (“свет–темнота”), то старт таймеров Т1, Т2;

если на Фэл2 перепад 0-1 стоп таймера Т1;

по перепаду Фэл1 1-0 (“темнота–свет ”) стоп таймера Т2;

рассчитать длину по формуле (6).

Отметим: если во время работы, когда бревно проходит мимо и загораживает один из фотоэлементов, конвейер остановится, то будет замерена неверная неправдоподобно большая длина, возможно, будет переполнение таймеров. Аналогично, если в момент включения питания МП системы бревно уже лежит на конвейере, то его длина не может быть измерена. Во избежание этих ситуаций надо: сделать невозможной остановку конвейера при включенной МП системе; в алгоритм самотестирования (POST) добавить проверку фотоэлементов на готовность (удостовериться, что их светоприемные части освещены).

Механизм отбраковки рассмотрим менее подробно. МП система должна лишь выдать сигнал логической “1” для бракованного бревна и “0” для нормального. Сигнал выдается в момент, когда задняя кромка бревна минует Фэл2 (по перепаду Фэл2 1-0). Далее он усиливается и подается на привод отбраковки. Привод (не вдаваясь в подробности) при наличии “1” на его входе поднимает вверх постоянно движущийся поперечный конвейер (➃ на рис. 2), и он сбрасывает бревно с основного конвейера.

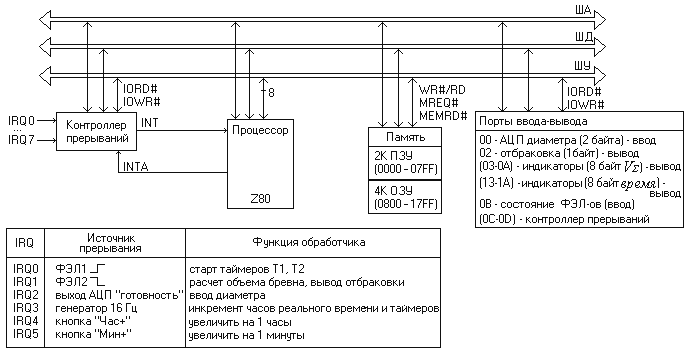
**Аппаратная часть микропроцессорной системы**

Общие характеристики

## Общая направленность предлагаемой конструкции системы характеризуется тем, что система ориентирована на работу с памятью, портами ввода-вывода и прерываниями. Подсистема прямого доступа у памяти вообще не используется, т.к. в ее использовании не возникло серьезной необходимости: при работе управляющей программы нет такого внешнего устройства, которое бы вело интенсивный обмен данными с памятью.

Структура системы приведена на рис. 5.

Рис.5 Структура микропроцессорной системы



### Ниже следуют пояснения по каждому блоку рис.5.

Блок центрального процессора, память и шины системы

“Сердцем” всей МП системы является микропроцессор (МП) Z80 (или его отечественный аналог – КР1856ВМ1).

Вкратце опишем используемые в МП системе выводы микропроцессора.

Сигналы, которые активны в нуле, будем отмечать суффиксом “#”.

* CLK – вход тактовой частоты 4 МГц;
* RESET# – вход сброса (активен в нуле);
* MREQ# – выход запроса памяти. Нулевое состояние свидетельствует, что микропроцессор обращается к памяти для чтения или записи;
* IORQ# – выход запроса портов ввода/вывода. Нулевое состояние означает, что идет обращение к портам;
* RD# – чтение. Активен, когда микропроцессор инициирует операцию чтения из памяти или из устройства ввода/вывода;
* WR# – запись. То же, что и RD#, но для операции записи;
* A0-A15 – шина адреса. Имеет 16 линий и используется для адресации к максимум 65535 ячейкам памяти и устройствам ввода-вывода;
* D0-D7 – шина данных. Восемь двунаправленных линий, используемых, для ввода/вывода данных микропроцессора, памяти и периферийных устройств.
* INT# – вход прерывания. Нуль на этом входе означает наличие запроса на прерывание от внешнего устройства.
* M1# – выход первого машинного цикла. Нуль на этом выходе показывает, что микропроцессор производит выборку кода операции. В нашей системе этот выход совместно с IORQ# служит для подтверждения прерывания.

Тактовым генератором служит мультивибратор, собранный на двух инверторах. Времязадающими цепями являются конденсатор, сопротивления и резонатор. Частота генератора 16 МГц (как и резонатора Z1). Затем эта частота двумя триггерами делится на 4, получается тактовая частота 4 МГц, которая и подается на вход CLK. Деление частоты нужно для большей ее точности.

Кнопка “RESET” в нажатом состоянии подает на соответствующий вход микропроцессора сигнал сброса. Для устранения эффекта “дребезга контактов” сигнал RESET# подается через D-триггер, записываясь в него по фронту первого же тактового импульса с линии CLK.

Для организации шин адреса (ША), данных (ШД), управления (ШУ) микропроцессорной системы к соответствующим выходам микропроцессора подключены буферные элементы. Их нагрузочная способность гораздо больше нагрузочной способности выводов микропроцессора. Они являются повторителями и “цифровыми усилителями” сигналов микропроцессора. Для ША это два 8-разрядных регистра ИР82. Для ШД – специальная микросхема – 8-разрядный двунаправленный шинный формирователь с возможным Z-состоянием выводов.

Четыре сигнала управления (WR#, RD#, MREQ#, IORQ#) сначала подвергаются дешифрации. Объединенные по схеме ИЛИ сигналы RD# и MREQ# дают в результате сигнал MEMRD# чтения памяти. Аналогично получены сигналы чтения порта IORD# и записи в порт IOWR#. Еще получен сигнал WR#/RD, который равен 0, когда активен сигнал WR# и равен 1, когда активен RD#. Пять сигналов (MEMRD#, MREQ#, WR#/RD, IORD#, IOWR#) понадобятся при организации обращения к памяти и портам. Все они объединены в 5-разрядную шину управления (ШУ), которая также имеет шинный формирователь.

Рассмотрим подключение микросхем памяти. Всего подключено 2К ПЗУ на одной микросхеме К541РЕ1 (адресное пространство 0000H – 07FFH) и 4К ОЗУ на двух микросхемах К537РУ8 (адреса 0800H – 17FFH). Принцип работы микропроцессора с памятью следующий. В первом такте машинного цикла обращения к памяти (чтения или записи) процессор выдает на ША адрес запрашиваемой ячейки. Этот адрес дешифруется так, что вход CS (выбор кристалла) соответствующей адресу микросхемы становится активным (нулем), сигнализируя, что эта микросхема выбрана для операции. Во втором такте МП формирует на шине управления сигналы управления. Сигнал MEMRD# подается на вход разрешения операции микросхемы ПЗУ. Сигнал WR#/RD подается на одноименные входы микросхем ОЗУ, а сигнал MREQ# – на вход разрешения операции микросхем ОЗУ. Итак, чтение/запись микросхемы памяти происходит, если одновременно выполнены два условия: микросхема выбрана адресом и ей разрешена операция сигналами управления.

Порты ввода-вывода

Всего МП система имеет 14 байт портов ввода и вывода. На рис. 5 можно видеть их адреса в пространстве ввода-вывода.

Механизм действия портов аналогичен действию микросхем памяти. Адрес порта на ША дешифруется в двенадцать сигналов выбора кристалла CS0#–CS13#, а каждый порт имеет возможность управляемого чтения/записи. Для порта ввода (порт диаметра) одновременно активные CS*i*# и IORD# стробируют выдачу одного байта на шину данных. Для порта вывода одновременно активные CS*i*# и IOWR# стробируют запись байта с шины данных в регистр порта.

Адреса портов ввода-вывода дешифруются отдельным дешифратором ИД3 с четырьмя входами и 16-ю выходами (первые 14 из которых – CS0#–CS13#). На входы подается младшая тетрада шины адреса – номер адресуемого порта, на всех выходах дешифратора образуются единицы, кроме одного нулевого выхода, имеющего этот номер.

Два адресам контроллера прерываний (0C-0D) дешифруются в один сигнал выборки кристалла, при этом контроллер принимает бит A0 адреса и самостоятельно разбирается, к какому из двух адресов идет обращение.

Порт ввода диаметра. Двухбайтный порт, выбирается адресами 00 и 01 (сигналами CS0# и CS1#). Датчик диаметра бревна является аналоговым. Чтобы ввести числовое значение диаметра в МП и использовать его в вычислениях, надо преобразовать напряжение с датчика в цифровой код. Этим занимается АЦП на специальной микросхеме К1113ПВ1А. Это 10-разрядный АЦП с выходными цифровыми уровнями ТТЛ с возможностью преобразования однополярного аналогового сигнала в диапазоне [0..10,5]В или биполярного сигнала в диапазоне [-5,5..5,5]В. Мы используем АЦП в однополярном режиме, поэтому вход 15 (Z1) присоединен к массе. Преобразуемое аналоговое напряжение подается на вход 13 (вход A). По заднему фронту (спаду) импульса на входе 11 (RES) АЦП сбрасывается и начинает преобразование. Преобразование занимает некоторое время (около 30 мкс). По окончании микросхема АЦП на выходах D0–D9 выводит итоговый цифровой код, а на выходе готовности данных 17 (RDY) выставляет логический 0, сигнализирующий “преобразование окончено”. На этом работа АЦП заканчивается.

#### Далее в МП системе сигнал с выхода АЦП RDY несет две функции: стробирует запись кода диаметра с АЦП в два буферных регистра ИР82 и посылает сигнал запроса прерывания на вход IR2 контроллера прерываний.

#### На вход RES АЦП мы подаем импульсы частотой Ѕ Гц, соответственно, раз в две секунды возникает прерывание IRQ2. Обработчик этого прерывания, при условии, что бревно сейчас под пластиной датчика (эта информацию можно прочесть из порта статуса фотоэлементов, описанного ниже) будет читать регистры диаметра в некий массив диаметров в ОЗУ. По прохождении бревна, когда запускается вычисление объема бревна, все введенные диаметры усредняются и в вычислениях используется среднее значение всех мгновенных диаметров.

Порт вывода сигнала отбраковки (адрес 02). Представляет собой обычный D-триггер. По одновременным сигналам CS2# и IOWR# в триггер записывается младший бит D0 шины адреса. Программа устанавливает D0=1, если бревно бракованное, иначе D0=0.

Порт вывода на индикаторы (8 байт). В эти ячейки программа заносит семисегментный код, выводимый на индикаторы. Сигналы выборки кристаллов индикаторов в схеме – CS3# (самый левый индикатор)…CS10# (самый правый).

#### Есть одна особенность индикации. На индикаторы возможен вывод объема *VΣ* или текущего времени – должно зависеть от кнопок управления индикацией на панели оператора. Пусть эти величины в семисегментном коде хранятся в разных ячейках памяти (каждая по 8 байт). Процессор, выполняя управляющую программу, то и дело копирует их содержимое в порт. Ячейку *VΣ* – в порт (03..0A), а ячейку времени – в порт (13..1A).

#### Как говорилось ранее, при обращении к портам дешифруются лишь младшие 4 бита с шины адреса. Поэтому и в том, и в другом случае обращение идет к одним и тем же физическим индикаторам. Дополнительной дешифрации, зависимой от кнопок управления индикации, подвергается бит A4 адреса. Если A4=1, значит, процессор пытается индицировать время, при этом нажатая кнопка “*время*” разрешает прохождение сигналов выборки кристалла. А в отжатом положении сигналы выборки не пройдут, и индикация не состоится. Аналогично нажатая кнопка “*VΣ*” разрешает адресацию к индикаторам при A4=0.

#### Этот вариант “двойной адресации” свою относительную сложность искупает экономией лишних 8-ми регистров, которые понадобились бы, если делать отдельные порты вывода для *VΣ* и для времени.

Порт ввода статуса фотоэлементов. Этот порт (адрес 0B) необходим для использования при первоначальном тестировании, когда система проверяет готовность фотоэлементов (контролирует выходы Фэл1 и Фэл2 на ноль). Во время работы содержимое порта несет смысл статуса обработки и помогает разобраться, на какой стадии своего движения находится бревно.

#### В порте действительно несут информацию только 2 бита: D0 – о состоянии Фэл1 (ноль на выходе Фэл1 или единица), D1 – о статусе Фэл2. Ноль свидетельствует об успешном прохождении света от лампы к светоприемнику фотоэлемента, единица – о том, что свет не проходит.

Порт контроллера прерываний (адреса 0C и 0D). Используется для адресации регистров контроллера: сигналы CS12# и CS13#, объединенные по ИЛИ, подаются на его вход выбора кристалла.

**Подсистема прерываний**

МП система должна периодически опрашивать датчики и оперативно реагировать на изменение их состояния. Один из вариантов опроса – МП в цикле управляющей программы опрашивает все датчики, анализирует их состояние; инициатива при вводе-выводе информации принадлежит микропроцессору. Второй вариант – опрос по прерываниям: само внешнее устройство, приготовив данные, подает к МП сигнал прерывания и тот считывает эти данные, выполняя процедуру обработки прерывания. Очевидно, что для нашей системы второй вариант преимуществен: медленно меняющийся диаметр лучше считывать по внешней периодической синхронизации (например, Ѕ Гц), чем в каждом цикле управляющей программы. Сигналы с фотоэлементов (точнее, их перепады, после которых надо производить старт/стоп таймеров), которые похожи на “исключительные ситуации”, также логичнее обрабатывать по прерываниям. Наконец, прерывания помогают организовать программно часы реального времени и таймеры T1 и T2, экономя на микросхемах. Недостаток только один: приходится усложнять систему, вводя программируемый контроллер прерываний (ПКП).

Опишем прерывания в МП системе.

1. IRQ0. Прерывание возникает по перепаду 0-1 на выходе фотоэлемента Фэл1. Обработчик должен сбросить и запустить таймеры T1 и T2.
2. IRQ1. Возникает по перепаду 1-0 на Фэл2 и означает, что измерения закончены, бревно полностью вышло из области действия датчиков. Обработчик должен рассчитать объем бревна *Vi*, суммарный объем *VΣ* и сформировать сигнал отбраковки.
3. IRQ2. Появляется по сигналу готовности данных на АЦП. Обработчик вводит 2 байта диаметра.
4. IRQ3. Источником служит генератор на 16 Гц. Обработчик делает инкремент часов реального времени и условный инкремент таймеров (анализируя информацию статуса):

|  |  |
| --- | --- |
|  | Если слово статуса ФЭЛ равно: |
| Инкремент T1 | xxxxxx01 |
| Инкремент T2 | xxxxxx01 или xxxxxx11 |
| Нет инкремента | иначе (xxxxxx00 или xxxxxx10) |

1. IRQ4 и 6) IRQ5. Сигналы приходят от кнопок “+Час” и “+Мин” на панели оператора. Обработчик незамедлительно инкрементирует часы (минуты) реального времени.
2. Казалось бы, перепады на выходах фотоэлементов равноценны, и перепады сигналов 1-0 на Фэл1 и 0-1 на Фэл2 стоило бы тоже обрабатывать по прерываниям. Но это не нужно. Обработчики этих прерываний несли бы одну функцию: стоп того или другого таймера. А по прерыванию IRQ3 делается *условный* инкремент таймеров, т.е. таймер не надо останавливать, он остановится сам собой, когда сигналы с фотоэлементов придут в определенное сочетание (см. таблицу).

Сигналы запросов IRQ*i* приходят на входы IR0–IR5 программируемого контроллера прерываний (запросы принимаются по переходу 0→1). Используется режим работы ПКП с фиксированными приоритетами прерываний. Максимальный приоритет имеет запрос IR0, минимальный – запрос IR5. На этом распределении приоритетов, кстати, основан выбор номеров для прерываний: чтобы прерывания, функционально более важные (от фотоэлементов и АЦП) имели больший приоритет, чем прерывания, несущие декоративный смысл (от часов и кнопок управления часами).

По выходу INT контроллер передает процессору запрос на прерывание. Процессор формирует сигналы подтверждения прерывания, приходящие на вход INTA контроллера.

Рассмотрим подключение остальных выводов. Контроллер будет работать в одиночном (не каскадном) режиме – это контролирует вывод SP=0 микросхемы.

На вход выбора кристалла CS подается дешифрованный адрес. На вход A0 подается нулевой разряд адреса. Выводы каскадирования CAS0–CAS2 остаются неподключенными. Выводы D0–D7 подключены к шине данных.

Перед работой ПКП нужно инициализировать. Управляющая программа делает это во время самотестирования системы. При инициализации нужно сообщить контроллеру адреса процедур обработки прерываний.

**Программная часть микропроцессорной системы**

Тестирующие программы

Тестовые программы запускаются сразу по включении питания микропроцессорной системы перед началом работы. Это тест ОЗУ, ПЗУ, тест фотоэлементов, инициализация программируемого контроллера прерываний, инициализация переменных системы.

Тест ОЗУ

Тест ОЗУ состоит в проверке всех ячеек ОЗУ на запись и верное считывание байтов 00Н и FFН. Если для какой-то ячейки совпадение записанного и прочитанного значений не состоялось, происходит переход на подпрограмму E\_RAM, в регистровой паре HL при этом адрес неисправной ячейки.

Листинг 1: тест ОЗУ

; – – – тест ОЗУ (адреса 0800…17FF)

TST\_RAM LD HL, 0800 ; HL←начало области ОЗУ

TST\_RAM1 XOR A ; A=0

LD (HL), A ; записать 0 в ячейку памяти

LD A, (HL) ; прочитать эту ячейку

OR A ;

JP NZ, E\_RAM ; если в ячейке не ноль, то выход

CPL ; инвертировать A (A=FF)

LD (HL), A ; записать FF в ячейку памяти

LD A, (HL) ; прочитать эту ячейку

INC A ; если прочитано A=FF, то A=FF+1=0

JP NZ, E\_RAM ; иначе выход

INC HL ; к следующему адресу

LD A, H ; проверить на достижение конца ОЗУ

CP #18 ;

JR NZ, TST\_RAM1 ; конец цикла

… ; тест ОЗУ успешен, продолжение тестов

Подпрограмма E\_RAM – это бесконечный цикл вывода на индикаторы признака ошибки ОЗУ (строки “Е1”) и адреса неисправной ячейки (например, “Е1 12AF”). Естественно, при этом дальнейшее выполнение управляющей программы и функционирование системы отменяется (поэтому переход на E\_RAM выполняется командой JP, а не CALL). По адресу можно определить, какая именно (из двух) микросхема памяти подлежит замене.

Независимый от кнопок управления индикацией вывод на индикаторы семисегментным кодом производится через 8-байтный порт вывода 03..0A и через порт 13..1A одновременно (см. п.3.3). Адрес 0A (он же 1A) соответствует самому левому индикатору, адрес 03 (он же 13) – самому правому.

Декодирование из двоичного в семисегментный код делается с помощью массива перекодировки длиной 16 байт, расположенного в ПЗУ по начальному адресу 0700Н:

0700 3F 06 5B 4F 66 6D 7D 07 ; “0”–“7”

0708 7F 6F 77 7C 39 5E 79 71 ; “8”–“F”

Для декодирования шестнадцатиричной цифры в семисегментный код достаточно извлечь из массива элемент с индексом, равным этой цифре.

Важно то, что процедура E\_RAM не обращается к памяти ни напрямую, ни через стек – она хранит все свои данные в регистрах и записывает только в порты индикаторов.

Листинг 2: процедура E\_RAM, вызываемая после ошибки ОЗУ

; – – – обработка ошибки ОЗУ (код “E1”)

E\_RAM EX DE, HL ; адрес-операнд из HL перенести в DE

LD H, #07 ; H←старший байт адреса таблицы кодов

LD A, E ;

AND #0F ; выделяем младшую тетраду в E

LD L, A ; извлекаем элемент с этим номером из массива

LD C, (HL) ; в регистр C

; теперь C=код, посылаемый на 8-й индикатор

LD A, D ;

AND #0F ; выделяем младшую тетраду в D

LD L, A ;

LD B, (HL) ; теперь B=код, выводимый на 6-й индикатор

SRL D ; четырежды сдвигаем D вправо (0→D→CY)

SRL D ; заполняя старшие биты нулями

SRL D ;

SRL D ;

SRL E ; четырежды сдвигаем E вправо (0→D→CY)

SRL E ; заполняя старшие биты нулями

SRL E ;

SRL E ;

LD L, E ; теперь необходимости в наложении маски нет

LD E, (HL) ; E=код, выводимый на 7-й индикатор

LD L, D ;

LD D, (HL) ; D=код, выводимый на 5-й индикатор

; цикл вывода на индикаторы

E\_RAM1 LD A, #79; A←семисегментный код символа “E”

OUT (#0A), A ; в 1-й слева индикатор

OUT (#1A), A ;

LD A, #06 ; A←“1”

OUT (#09), A ; во 2-й индикатор

OUT (#19), A ;

XOR A ; A←“ ” (пусто)

OUT (#08), A ; в 3-й

OUT (#18), A ;

OUT (#07), A ; в 4-й

OUT (#17), A ;

LD A, D ; выводим DBEC на индикаторы 5678

OUT (#06), A ;

OUT (#16), A ;

LD A, B ;

OUT (#05), A ;

OUT (#15), A ;

LD A, E ;

OUT (#04), A ;

OUT (#14), A ;

LD A, C ;

OUT (#03), A ;

OUT (#13), A ;

JR E\_RAM1 ; переход на начало цикла

**Тест ПЗУ**

Тест ПЗУ состоит в вычислении контрольной суммы ПЗУ, т.е. суммы всех байтов в ПЗУ без учета переполнения суммы, и сравнении полученной суммы с известным эталонным значением. Несовпадение свидетельствует о явной неисправности ПЗУ, совпадение не обязательно говорит об его исправности (могут быть, хотя маловероятно, две компенсирующие друг друга ошибки).

Для этого теста (чтобы знать эталон, с которым сравнивать) надо иметь все подпрограммы ПЗУ. Однако сейчас мы только на стадии заполнения постоянной памяти. Предлагается следующий путь: в составленной ниже программе теста ПЗУ вычисляем контрольную сумму и сравниваем ее с нулем. А после того, как запрограммируем ПЗУ полностью, последнюю ячейку ПЗУ (17FFН) оставим свободной и занесем в нее число, дополняющее сумму всего ПЗУ без последней ячейки до нуля.

Получив в сумме всех ячеек ПЗУ не ноль, программа тестирования вызывает зацикленную подпрограмму E\_ROM, имея в регистре E число 2, характеризующее код ошибки.

Листинг 3: Тест ПЗУ

; – – – вычисление контрольной суммы ПЗУ

TST\_ROM XOR A ; A=0

LD H, A ;

LD L, A ; HL=0

TST\_ROM1 ADD A, (HL) ; A←A+(HL)

INC HL ; увеличить адрес ПЗУ на 1

LD B, A ; сохранить сумму A на время

LD A, H ; и проверить, достигли ли конца ПЗУ

CP #08 ;

LD A, B ; восстановить сумму A

JR NZ, TST\_ROM1;

OR A ; если конец ПЗУ, проверить A на ноль

LD E, 2 ;

JP NZ, E\_ROM ;если не ноль, то ошибка ПЗУ

… ; иначе тест ПЗУ успешен, продолжаем тесты

Подпрограмма E\_ROM выводит на два самых левых индикатора условный код ошибки ПЗУ (“E2”), а остальные индикаторы гасит. Индикация такой ошибки говорит о необходимости проверки и перепрограммирования микросхемы ПЗУ.

Листинг 4: процедура E\_ROM, вызываемая после ошибки ПЗУ

; – – – обработка ошибки ПЗУ (код “E2”)

E\_ROM LD A, #79 ; A←“E”

OUT (#0A), A ;

LD A, E ; A←код ошибки (2)

OUT (#09), A ;

XOR A ; A←“0”

LD C, #08 ;

LD D, #18 ;

LD B, 6 ; счетчик

E\_ROM1 OUT (C), A ; последние шесть индикаторов гасим

OUT (D), A ;

DEC C ;

DEC D ;

DJNZ E\_ROM1 ;

JR E\_ROM ;

**Тест фотоэлементов**

Перед началом работы нужно проверить прохождение света от ламп к фотоэлементам. Если один из фотоприемников не чувствует света, это значит, что или перегорела лампа фотоэлемента, или в момент пуска бревно уже лежит между фотоэлементами и его длина не может быть измерена, или повредился канал от фотоэлемента к процессору. В обоих случаях микропроцессорная система не начнет свою работу (т.к. нет возможности отличить второй случай от остальных). При обнаружении такой ошибки на индикаторы выводится признак ошибки фотоэлемента в виде условного кода “E3”.

В программе этого теста читается порт ввода статуса фотоэлементов (адрес 0B) и проверяются на ноль два его младших бита. Если хотя бы один не равен нулю, то происходит переход на обработчик ошибки. Причем для обработки ошибки можно использовать ту же процедуру E\_ROM, перед вызовом загрузив в регистр E код ошибки 3 (экономия памяти).

Листинг 5: тест фотоэлементов

; – – – проверка двух младших битов порта ФЭЛ на 0

###### TST\_FEL IN A, (#0B) ; прочесть слово статуса ФЭЛ

AND 3 ; наложить на него маску 000000112

LD E, 3 ;

JP NZ, E\_ROM ; если не ноль, то ошибка

… ; иначе продолжаем тесты

Инициализация программируемого контроллера прерываний

Для инициализации контроллера надо переслать ему два управляющих слова ICW1 и ICW2, первое по адресу 0CН (A0=0), второе – по адресу 0DН (A0=1).

Пересылаются следующие управляющие слова:

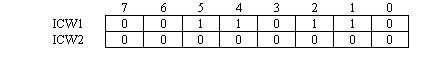


Рис. 6 Применяемые управляющие слова инициализации ПКП

Слово ICW1 установит одиночный (без каскадного соединения) режим работы ПКП (бит 1), 4-хбайтный интервал для начальных адресов обработчиков прерываний (бит 2).

Биты 5–7 слова ICW1 вместе со всем словом ICW2 сообщат контроллеру, что первый обработчик (запроса IR0) начинается с адреса 0020Н.

Контроллер накладывает определенные ограничения на расположение обработчиков в памяти. Первое из них в том, подпрограммы обработки прерываний должны располагаться по порядку, начиная с адреса обработчика запроса IR0, и с постоянным интервалом, т.е. образовывать в памяти таблицу. Интервал расположения может составлять 4 байта (если бит 2 ICW1 равен единице, как в нашем случае), или 8 байт (если бит 2 равен нулю). При постоянном интервале адреса всех обработчиков определятся расположением первого (IR0).

Адрес первого обработчика составляется из полного слова ICW2 (старший байт) и битов 7,6,5 слова ICW1 (старшие три бита младшего байта адреса).

В нашем случае старший байт равен 00Н, младший байт равен 0010 0000 = 20Н. Обработчики имеют начальные адреса: 0020H, 0024H, 0028H, 002CH, 0030H, 0034H (6 обработчиков).

Процедура INI\_PIC инициализирует контроллер.

Листинг 6: инициализация программируемого контроллера прерываний

; – – – переслать 38H в порт 0CH и 0 в порт 0DH

INI\_PIC LD A, #38 ;

OUT (#0C), A ;

XOR A ;

OUT (#0D), A ;

Инициализация переменных системы

В системе за некоторыми ячейками памяти закреплена функция хранения переменных. Например, подсчитанный суммарный объем древесины *VΣ* в двоичном формате хранится в отдельных двух байтах памяти с адресами V\_SUM, V\_SUM+1.

Требуют инициализации только две переменные (инициализируются нулем):

*VΣ* (2 байта) – начальный адрес V\_SUM;

*Время* (4 байта) – начальный адрес TIME.

Ясно, что эти имена (V\_SUM и TIME) лишь условные обозначения (также, как, например, имена меток в листингах программ). При переводе в машинный код эти имена транслируются в двухбайтный адрес.

Листинг 7: инициализация ячеек суммарного объема и времени

; – – – начальное обнуление объема и времени

INI\_VAR XOR A ;

LD (V\_SUM), A ;

LD (V\_SUM+1), A ;

LD (V\_SUM+2), A ;

LD (TIME), A ;

LD (TIME+1), A ;

LD (TIME+2), A ;

LD (TIME+3), A ;

… ; следует продолжение переходит к основному циклу работы (описание см. в п.4.4).

**Арифметические подпрограммы**

Микропроцессорной системе необходимо “уметь” выполнять арифметические операции: сложение, вычитание, умножение, деление и косинус. В этом же разделе приведем и неарифметические процедуры для осуществления индикации: преобразования данных в двоично-десятичный код и в семисегментный.

Основным форматом чисел в МП системе является двухбайтный формат с фиксированной точкой вида 1байт , 1байт . Формат беззнаковый, предполагается, что числа положительны. Одно число умещается в одной регистровой паре. Минимальное представимое число – 0,01H=1/256=3,9⋅10-3, максимальное – FF,FFН=256,996.

Общие правила для всех вычислительных процедур:

* программист вызывающей программы сам следит, чтобы операнды и результат не выходили за пределы представления и чтобы при вычитании не образовалось отрицательных величин;
* операнды при сложении, вычитании, умножении, делении хранятся в регистровых парах HL и DE, результат в HL. Процедура косинуса получает операнд и выводит результат в паре HL.

Сложение

В микропроцессоре Z80 есть команда сложения регистровых пар ADD HL, DE. Однако, чтобы сохранить единообразие, оформим ее все же как подпрограмму.

Листинг 8: подпрограмма PLUS

; – – – подпрограмма сложения

; HL+DE→HL

; сохраняет A, BC, DE

PLUS ADD HL, DE ;

RET ;

Вычитание

В Z80 есть команда вычитания регистровых пар с учетом переноса SBC HL, DE. Используем ее в подпрограмме вычитания, обнулив прежде флаг CY.

Листинг 9: подпрограмма MINUS

; – – – подпрограмма сложения

; HL–DE→HL

; сохраняет A, BC, DE

MINUS OR A ; A не меняется, но флаг CY=0

SBC HL, DE ;

RET ;

Умножение

При умножении первый множитель хранится в паре BC (скопируем его туда из HL в начале), второй множитель хранится в DE. Результат первоначально получается в четырехбайтном виде HL.DE, затем “обрезается” до H.L.

Результат произведения накапливается в HLDE. В течение 16 циклов сдвигов HLDE второй множитель DE постепенно выдвигаясь, уходит вправо и на его место приходит из HL готовые биты произведения. Они получаются при суммировании первого множителя (BC) и левой половины накопленной суммы HLDE (HL). Суммирование делается в случае, если перед этим при сдвиге HLDE вправо из DE был выдвинут единичный бит.

Листинг 10: подпрограмма MUL

; – – – подпрограмма умножения

; HL×DE→HL

; все регистры меняются

MUL LD B, H ; BC=1-й множитель

LD C, L ;

LD HL, 0 ; HL=0

LD A, 16 ; счетчик цикла

MUL2 SRL H ; сдвиг 0→HLDE→CY

RR L ;

RR D ;

RR E ;

JR NC, MUL1 ; если выдвинут 0, то на конец цикла

ADD HL, BC ; если выдвинута 1, то сложить

MUL1 DEC A ;

JR NZ, MUL2 ;

LD H, L ; получено произведение HL.DE

LD L, D ; которое “обрезаем” до L.D и переносим в H.L

RET

Деление

При делении делимое хранится в HL, делитель в DE. Результат получается в виде трех байт [стек.L] (где “стек” – содержимое вершины стека), в конце программы младший байт стека переносится в H и окончательный результат имеет формат H.L. Регистровая пара BC в начале обнуляется.

Деление производится в течение 24-х циклов. В каждом цикле делается сдвиг BC←HL←0, т.е. делимое HL выдвигается влево в пару BC. Затем BC сравнивается с DE (путем вычитания BC–DE и проверки флага переноса). Если BC>DE, то младший разряд HL устанавливается единицей, в противном случае он остается нулем. Это один из разрядов частного.

После 16 сдвигов в делимое HL полностью сдвинется влево и на его место придет частное целочисленного деления, в BC будет остаток деления, DE сохранит делитель. На этом этапе целая часть частного в HL заносится в стек, а HL обнуляется. В оставшихся 8-ми сдвигах BC←HL в BC будут сдвигаться только нули. После всех 24-х циклов в регистре L будет дробная часть частного, а в стеке целая часть.

Листинг 11: подпрограмма DIV

; – – – подпрограмма деления

; HL/DE→HL

; сохраняет DE, использует стек (2 байта)

DIV LD BC, 0 ;

LD A, 24 ; А – счетчик циклов

DIV4 PUSH AF ; сохранить счетчик А в стеке

ADD HL, HL ; BC←HL←0

RL C ;

RL B ;

BC

DE

BC

LD A, C ;

SUB E ;

LD C, A ;

LD A, B ;

SBC D ;

LD B, A ;

JR NC, DIV1 ; если BC>DE, то переход

LD A, C ;

BC

DE

BC

ADD E ;

LD C, A ;

LD A, B ;

ADC D ;

LD B, A ;

JR DIV2 ;

DIV1 INC HL ; если BC>DE: установить последний бит

; частного в 1

DIV2 POP AF ; извлечь счетчик А из стека

CP #09 ; проверить, 16-й цикл идет или нет

JR NZ, DIV3 ;

PUSH HL ; если 16-й цикл, то в HL целая часть частного,

; сохранить ее в стеке, чтобы извлечь

; в конце программы

LD HL, 0 ;

DIV3 DEC A ;

JR NZ, DIV4 ; конец цикла

POP BC ; извлекаем из стека целую часть частного

LD H, C ; получаем частное в H.L

RET ;

Косинус

Чтобы вычислить косинус, используется ограниченный четырьмя членами ряд Тейлора:

, угол α изменяется от 0 до π/2. (6)

Эта формула дает косинус с максимальной погрешностью 8,9⋅10-4, это меньше, чем ошибка разрядности используемого формата [1].[1] дробных чисел.

 (7)

Представление (7) формулы (6) гораздо удобнее для вычисления косинуса программно. Достаточно сделать процедуру для вычисления , где *n* – целое число, *β* – дробное число. Перед обращением к этой процедуре (названной COS\_A) операнды содержатся:

HL = α2;

DE = *β*;

BC = *n* в формате *n*.0 (B=*n*, C=0).

Результат – в HL

Эта подпрограмма использует предыдущие арифметические программы.

Листинг 12: подпрограмма COS\_A

; – – – вспомогательная подпрограмма для косинуса

; операнды: HL, DE, BC

; изменяются все регистры

COS\_A PUSH BC ; сохранить в стеке *n* перед вызовом MUL

CALL MUL ; HL=HL×DE

POP DE ; DE=*n*

CALL DIV ; HL=HL/DE

LD D, 1 ; DE=1.0

LD E, 0 ;

EX HL, DE ;

CALL MINUS ; HL=HL-DE

RET ;

Подпрограмма косинуса вычисляет косинус угла в паре HL.

Листинг 13: подпрограмма COS

; – – –подпрограмма косинуса

; операнд и результат в HL

; изменяются все регистры

COS LD D, H ;

LD E, L ;

CALL MUL ; HL=α2

PUSH HL ; α2 в стеке

LD D, 1 ; DE=1.0

LD E, 0 ;

LD B, #1E ; BC=1E.0H=30

LD C, 0 ;

CALL COS\_A ; HL=1-α2/30

EX HL, DE ;

POP HL ; HL=α2

PUSH HL ; α2 в стеке

LD B, #0C ; BC=C.0H=12

LD C, 0 ;

CALL COS\_A ; HL=1-α2/12(1-α2/30)

EX HL, DE ;

POP HL ; HL=α2

LD B, #02 ; BC=2.0

LD C, 0 ;

CALL COS\_A ; HL=1-α2/2[1-α2/12(1-α2/30)]

RET ;

**Преобразование двоичный→двоично-десятичный код**

В управляющей программе надо переводить двухбайтные числа формата [12].[34] в двоично-десятичный код вида [123.45] размером в пять тетрад для последующего перевода в семисегментный код и индикации.

Для этого нужны две отдельных подпрограммы: одна (по имени B2D) для перевода целой части числа [12] и вторая (имя B2D\_F) для перевода дробной части числа [34].

Подпрограмма B2D переводит целое двоичное число в регистре C (0..FFH) в 2-10 код, расположенный в регистровой паре HL. Перевод производится в соответствии с формулой:

HL=(…((*c*7)⋅2+*c*6)⋅2+…+*c*1)⋅2+*c*0,

в которой *ci* – разряды числа в регистре C, а удвоение и сложение с битами *ci* происходит по правилам десятичной арифметики (с командой DAA после операции).

Листинг 14: подпрограмма B2D

; – – – перевод байта (целого) в 2-10 код

; операнд C – переводимое число, результат в HL

; сохраняет DE

B2D LD B, 8 ;

B2D1 SLA C ; CY←C (получаем последний бит операнда)

LD A, L ; удвоение HL с учетом переноса CY

ADC L ; по правилам десятичной арифметики

DAA ;

LD L, A ;

LD A, H ;

ADC H ;

DAA ;

LD H, A ;

DJNZ B2D1 ; конец цикла

RET ;

Подпрограмма B2D\_F переводит дробное число в формате 0.L в 2-10 код из трех тетрад в формате 0.ABC (учитываются три цифры после запятой). Регистры A, B, C содержат в конце каждый по одной десятичной цифре. Перевод происходит так. Число 0.L умножается на 10, результат в паре H.L. Его целая часть (H) и будет первой цифрой A результата. Затем H обнуляется, полученная дробь 0.L снова умножается на 10 и т.д.

Для быстрого умножения на 10 сделана отдельная подпрограмма MUL10, умножающая пару HL (где H=0) и получающая результат в той же HL. Она использует равенство:

10⋅HL=2⋅HL+8⋅HL, а умножения на 2 и на 8 делаются с помощью команды ADD HL, HL.

Листинг 15: подпрограммы B2D\_F и MUL10

; – – – перевод байта (дробного) в 2-10 код

; операнд C – число с фиксированной перед старшим разрядом точкой, результат в ABC

; изменяются все регистры

B2D\_F LD H, 0 ;

CALL MUL10 ; получить в H первую цифру

LD A, H ; скопировать ее в A

LD H, 0 ; и обнулить H

CALL MUL10 ;

LD B, H ; вторую цифру – в регистр B

LD H, 0 ;

CALL MUL10 ; третью – в C

LD C, H ;

RET

; – – – умножение на 10

; операнд в HL (имеет значение только L) и результат в HL

; сохраняет A, BC

MUL10 ADD HL, HL ; HL⋅10=HL⋅2+HL⋅8

LD D, H ;

LD E, L ;

ADD HL, HL ;

ADD HL, HL ;

ADD HL, DE ;

RET ;

Преобразование двоичный→семисегментный код

Это преобразование с помощью таблицы перекодировки уже встречалось в тестовой программе ОЗУ. Сейчас оформим ее как отдельную подпрограмму (в тесте ОЗУ нельзя вызывать подпрограммы, т.к. команда вызова CALL использует стек). Операнд подпрограммы – двоично-десятичная цифра в регистре A (0..9, старшая тетрада нулевая).

Листинг 16: подпрограмма D27

; – – – перевод байта в семисегментный код

; операнд (0..9) и результат в A

; сохраняет BC, DE

D27 LD H, #07 ;

LD L, A ;

LD A, (HL) ;

RET ;

**Обработчики прерываний**

Всего их шесть – по числу прерываний. Обработчик обязательно должен сохранить в стеке все изменяемые им регистры и в конце восстановить их. Выход из обработчика выполняется не стандартным RET, а командой RETI.

**Обработчик IRQ0 (начало измерений)**

Функция обработчика – обнулить таймеры T1 и T2, а также специальную ячейку памяти D\_NUM (2 байта). Эта ячейка инкрементируется всякий раз после чтения напряжения *Ud* с датчика диаметра. По приходу запроса IRQ0, когда приходит новое бревно, она должна быть обнулена.

Листинг 17: обработчик запроса IRQ0

; – – – обнулить T1, T2, D\_NUM

IR0\_H PUSH BC ;

PUSH AF ;

LD BC, 0 ;

LD (T1), BC ; обнулить T1

LD (T2), BC ; обнулить T2

LD (D\_NUM), BC ; обнулить D\_NUM

POP AF ;

POP BC ;

RETI ;

**Обработчик IRQ2 (информация с АЦП готова)**

Функция обработчика – если бревно сейчас под пластиной датчика (можно судить, прочтя порт фотоэлементов с адресом 0B), считать два байта напряжения из портов 00 (младший) и 01 (старший). Записать их в массив напряжений, под который отведена область памяти начиная с адреса 0100Н до конца ОЗУ (всего 2К). Перед записью проверить, не заполнен ли этот массив. Инкрементировать ячейку D\_NUM, содержащую число элементов этого массива.

Листинг 18: обработчик запроса IRQ2

; – – – считать и обработать байт с АЦП

IR2\_H PUSH BC ;

PUSH HL ;

PUSH AF ;

IN A, (#0B) ;

AND #02 ; наложить маску 000000010

JR Z, IR2\_H1 ; если второй бит нулевой, то выход

LD HL, #1000 ; рассчитать адрес очередного элемента массива

LD C, (D\_NUM) ; считать D\_NUM

LD B, (D\_NUM+1) ;

SLA C ; умножить его на 2

RL B ;

ADD HL, BC ; теперь адрес в HL

LD A, H ;

CP #18 ;

JR NC, IR2\_H1 ; если вышли за пределы массива, то выход

IN A, (#00) ; считать первый байт с АЦП

LD (HL), A ; и отправить его в память

INC HL ;

IN A, (#01) ; считать второй

LD (HL), A ; отправить

INC BC ; увеличить переменную D\_NUM на единицу

LD (D\_NUM), BC ;

IR2\_H1 POP AF ;

POP HL ;

POP BC ;

RETI ;

**Обработчик IRQ3 (от генератора 16 Гц)**

Функция обработчика – произвести инкремент часов реального времени и условный инкремент таймеров. Часы реального времени – это 4 байта в памяти:

TIME 1/16 секунды (0..15);

TIME+1 секунды (0..59);

TIME+2 минуты (0..59);

TIME+3 часы (0..23);

Все величины хранятся в двоичном формате.

Таймерам T1 и T2 отведено по 2 байта с начальными адресами T1 и T2. Условия, при которых они инкрементируются, были приведены в п. 3.4.

Листинг 19: обработчик запроса IRQ3

; – – – инкремент часов реального времени и условный инкремент таймеров

IR3\_H PUSH BC ;

PUSH HL ;

PUSH AF ;

; часы реального времени

LD HL, TIME ;

INC (HL) ; инкремент 1/16 секунд

LD A, (HL) ;

CP 16 ; проверить на достижение максимума JR C, IR3\_H1 ; условный выход из подпрограммы

LD (HL), 0 ; иначе обнулить 1/16 секунды и продолжить

INC HL ;

LD B, 2 ; инкремент секунд и минут делается в цикле

IR3\_H2 INC (HL) ;

LD A, (HL) ;

CP 60 ;

JR C, IR3\_H1 ;

LD (HL), 0 ;

INC HL ;

DJNZ IR3\_H2 ; конец цикла

INC (HL) ; инкремент часов

LD A, (HL) ;

CP 24 ;

JR C, IR3\_H1 ;

XOR A ; если счетчик часов=24

LD (HL), A ; то обнулить все 4 байта часов реального времени

DEC HL ;

LD (HL), A ;

DEC HL ;

LD (HL), A ;

DEC HL ;

LD (HL), A ;

; таймеры

IR3\_H1 IN A, (#0B) ; загрузить слово статуса фотоэлементов

AND 1 ;

JR Z, IR3\_H3 ; если не установлен 1-й бит, то выход

LD HL, T2 ; иначе инкремент Т2

INC (HL) ;

JR NZ, IR3\_H4 ; если инкремент не обнулил

; первый байт Т2, то идем дальше

INC HL ; иначе увеличить на 1 и второй байт

INC (HL) ;

IR3\_H4 IN A, (#0B) ;

AND 2 ; проверить 2-й бит статуса ФЭЛ

JR Z, IR3\_H3 ; если он не установлен, то выход

LD HL, T1 ; иначе инкремент Т1

INC (HL) ;

JR NZ, IR3\_H3 ;

INC HL ;

INC (HL) ;

IR3\_H3 POP AF ;

POP HL ;

POP BC ;

RETI ;

**Обработчик IRQ4 (от кнопки “+Час”)**

Функция обработчика – увеличить на единицу часы реального времени (ячейка TIME+3).

Листинг 20: обработчик запроса IRQ4

; – – – инкремент часов

IR4\_H PUSH HL ;

PUSH AF ;

LD HL, TIME+3;

INC (HL) ; инкремент часов

LD A, (HL) ;

CP 24 ;

JR C, IR4\_H1 ;

XOR A ; если счетчик часов=24

LD (HL), A ; то обнулить часы и минуты

DEC HL ;

LD (HL), A ;

IR4\_H1 POP AF ;

POP HL ;

RETI

**Обработчик IRQ5 (от кнопки “+Мин”)**

Функция обработчика – увеличить на единицу минуты реального времени (ячейка TIME+2).

Листинг 21: обработчик запроса IRQ5

; – – – инкремент минут

IR5\_H PUSH HL ;

PUSH AF ;

LD HL, TIME+2;

INC (HL) ; инкремент минут

LD A, (HL) ;

CP 60 ;

JR C, IR5\_H1 ;

XOR A ; если счетчик минут=60

LD (HL), A ; то обнуление минут

INC HL ; и инкремент часов

INC (HL) ;

LD A, (HL) ; с проверкой часов на 24

CP 24 ;

JR C, IR5\_H1 ;

XOR A ; если счетчик часов=24

LD (HL), A ; то обнулить и часы

IR4\_H1 POP AF ;

POP HL ;

RETI

**Обработчик IRQ1 (от фотоэлемента Фэл2)**

Обработчик IRQ1 выполняет самую важную функцию. Его задача – вычислить объем бревна. Последовательность следующая: вычисляем диаметр бревна, длину, вычисляем объем *Vi*, находим объем *VΣ*.

Для вычисления диаметра все значения, прежде считанные в массив напряжений с АЦП, усредняются: суммируются и делятся на количество (D\_NUM). При суммировании может произойти переполнение суммы (а она двухбайтная), чтобы этого не было, массив разбивается на группы по 16 измерений в каждой. Если осталась остаточная группа с числом меньше 16, то она отбрасывается. В каждой из них подсчитывается среднее, затем рассчитывается искомое как среднее средних.

Из среднего напряжения находится угол

α=.

Затем находим диаметр *d=*0,625 – 0,5cos α=00,A0H – 00,80Hcos α. Занести его в ячейку DIAM (2 байта в памяти). Сравнить диаметр с допустимыми пределами [0,2..0,5]=[0,33H..0,8H]. Если он выходит за эти пределы, то выдать на отбраковку (порт 02H) единицу.

Объем *Vi* находится как *Vi*=(*π*/4)*d*2⋅T1/T2=0,C9H⋅*d*2⋅T1/T2.

Листинг 22: обработчик запроса IRQ1

; – – – найти объем бревна и суммарный объем

; усреднение всех напряжений с датчика диаметра в массиве по адресу 1000H

IR1\_H PUSH AF ;

PUSH BC ;

PUSH DE ;

PUSH HL ;

LD L, (D\_NUM) ;

LD H, (D\_NUM+1) ;

LD B, 4 ; делим D\_NUM на 16

IR1\_H1 SRL H ;

RR L ;

DJNZ IR1\_H1 ;

LD C, L ; в результате C=число групп по 16

PUSH BC ; сохранить С в стеке

LD HL, #1000 ;

LD DE, 0 ; DE – начальная сумма групп

PUSH DE ; отправить ее в стек, C станет второй в стеке

PUSH DE ; DE – начальная сумма отдельной группы,

; отправить→в стек, сумма групп вторая в стеке

; C – третья в стеке

IR1\_H4 LD B, 16 ;

IR1\_H2 LD E, (HL) ; читаем в DE элемент массива

INC HL ;

LD D, (HL) ;

INC HL ;

EX (SP), HL ; текущую сумма в HL, текущий адрес в стеке

ADD HL, DE ;

EX (SP), HL ; новая сумма в стеке, текущий адрес в HL

DJNZ IR1\_H2 ;

; в итоге сумма одной группы по 16 – в стеке

; начальный адрес следующей группы – в HL

POP DE ;

LD B, 4 ; находим среднее одной группы,

IR1\_H3 SRL D ; деля сумму в DE на 16

RR E ;

DJNZ IR1\_H3 ;

POP HL ; берем из стека сумму групп

ADD HL, DE ;

PUSH HL ; снова отправляем в стек: сначала сумму групп

PUSH DE ; затем сумму одной группы

DEC C ; С – счетчик групп

JR NZ, IR1\_H4 ; следующая группа…

POP HL ;

POP HL ; извлечь найденную сумму групп

POP BC ; извлечь счетчик групп C

LD D, C ; DE=C.0

LD E, 0 ;

CALL DIV ; делим сумму групп на число групп

; теперь HL=Ud=среднее всего массива напряжений датчика

; следующий шаг – нахождение угла α, cos α, d

LD D, 4 ; DE=491H

LD E, #91 ;

LD B, H ; сохранить Ud в BC

LD C, L ;

EX DE, HL ;

CALL MINUS ; HL=491H-Ud

EX HL, DE ;

LD H, B ;

LD L, C ;

CALL DIV ; HL= Ud/(491H-Ud)

LD D, 0 ;

LD E, #4F ;

CALL MUL ; HL=α

CALL COS ; HL=cos α

LD D, 0 ;

LD E, #80 ;

CALL MUL ; HL=0,5cos α

LD D, 0 ;

LD E, #A0 ;

EX HL, DE ;

CALL MINUS ; HL=0,625-0,5cos α=d

LD (DIAM), HL ; занести диаметр в память

; формируем сигнал отбраковки

LD A, L ;

CP #33 ;

JR NC, IR1\_H5 ; если d>20, то идем дальше

LD A, 1 ; иначе в порт отбраковки записать 1

OUT (#02), A ;

JR IR1\_OUT ; и выход, не считая объем

IR1\_H5 LD A, L ;

CP #80 ;

JR C, IR1\_H6 ; если d<50, то идем дальше

LD A, 1 ; иначе в порт отбраковки записать 1

OUT (#02), A ;

; дальше находим Vi, VΣ

IR1\_H6 LD L, (T1) ;

LD H, (T1+1) ;

LD E, (T2) ;

LD D, (T2+1) ;

CALL DIV ; HL=T1/T2

LD E, (DIAM) ;

LD D, (DIAM+1) ;

PUSH DE ;

CALL MUL ; HL=T1⋅d/T2

POP DE ;

CALL MUL ; HL=T1⋅d2/T2

LD D, 0 ;

LD E, #C9 ;

CALL MUL ; HL=(π/4)T1⋅d2/T2=Vi

EX HL, DE ;

LD L, (V\_SUM) ;

LD H, (V\_SUM+1) ;

CALL PLUS ; HL=VΣ

LD (V\_SUM), HL ; занести суммарный объем в память

IR1\_OUT POP HL ;

POP DE ;

POP BC ;

POP AF ;

RETI ;

Основная исполняемая часть программы

Эту часть, которая, собственно, и является управляющей программой, микропроцессор выполняет большую часть своего времени, изредка отвлекаясь от нее на обработку прерываний.

Ее функции:

прочесть *VΣ* из ячейки V\_SUM, перевести его в 2-10 код (подпрограммы B2D, B2D\_F), перевести каждую цифру в семисегментный код (подпрограмма D27) и вывести в порт индикаторов (03-0A), присоединив десятичную запятую в индикаторе 06Н;

прочесть *время* из ячейки TIME (TIME+1–секунды, TIME+2–минуты, TIME+3–часы), перевести его в 2-10 код, перевести каждую цифру в семисегментный код и вывести в порт индикаторов (13-1A), присоединив десятичную запятую в индикаторах 13Н, 15Н, 17Н.

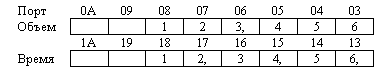


Рис. 7 Расположение объема и времени на индикаторах

Листинг 23: основная часть

; – – – вывести на индикаторы объем и время

; вывод объема

MAIN LD E, (V\_SUM) ;

LD D, (V\_SUM+1) ; D.E – суммарный объем

; выводим целую часть D

LD C, D ;

CALL B2D ; HL=2-10 код D

LD A, H ;

CALL D27 ; в семисегментный код

OUT (#08), A ; вывести в порт

LD A, H ;

AND #0F ;

CALL D27 ;

OR #80 ; примешать десятичную запятую

OUT (#06), A ; вывести в порт

LD A, H ;

RR A ; четырежды сдвигаем вправо

RR A ;

RR A ;

RR A ;

AND #0F ;

CALL D27 ;

OUT (#07), A ; вывести в порт

; выводим дробную часть E

LD L, E ;

CALL B2D\_F ; ABC=три цифры 2-10 кода числа 0.L

CALL D27 ;

OUT (#05), A ; вывести в порт

LD A, B ;

CALL D27 ;

OUT (#04), A ; вывести в порт

LD A, C ;

CALL D27 ;

OUT (#03), A ; вывести в порт

; вывод текущего времени

LD HL, (TIME+1) ; HL=адрес секунд

LD B, #13 ; самый правый индикатор

MAIN1 PUSH HL ;

LD A, (HL) ;

LD C, A ;

CALL B2D ; HL=2-10 код секунд/минут/часов (причем Н=0)

LD A, L ;

AND #0F ;

CALL D27 ;

OR #80 ; примешать запятую

OUT (B), A ; вывести в порт

INC B ; следующий индикатор

LD A, L ;

RR A ; четырежды сдвигаем А вправо

RR A ;

RR A ;

RR A ;

AND #0F ;

CALL D27 ;

OUT (B), A ; вывести в порт

INC B ; следующий индикатор

POP HL ;

INC HL ; перейти к следующей ячейке (TIME+2, TIME+3)

LD A, B ;

CP #08 ; проверка конца цикла

JR NZ, MAIN1 ;

JP MAIN ; начинаем все сначала

**Общая структура ПЗУ микропроцессорной системы**

# Наконец, приведем карты ПЗУ и ОЗУ микропроцессорной системы.

# ПЗУ

|  |  |
| --- | --- |
| 0000 | DI |
| 0001 | Начало тестов  TST\_RAM  JR 0038 |
| 0020 | Распределитель обработчиков прерываний  0020 JP IR\_H0  0024 JP IR\_H1  0028 JP IR\_H2  002C JP IR\_H3  0030 JP IR\_H4  0034 JP IR\_H5 |
| 0038 | Продолжение теста ОЗУ (TST\_RAM)  TST\_ROM  TST\_FEL  INI\_PIC  INI\_VAR |
|  | EI |
|  | LD SP, #0FFF |
|  | Главный цикл программы  MAIN |
|  | Подпрограммы  E\_RAM  E\_ROM  PLUS  MINUS  MUL  DIV  COS\_A  COS  B2D  B2D\_F  MUL10  D27 |
|  | Обработчики  IR\_H0  IR\_H1  IR\_H2  IR\_H3  IR\_H4  IR\_H5 |
|  | Область нулей 00000000 |
| 0700 | Таблица перекодировки |
| 070F | 00000000 |
| 07FF | Последний байт (дополнение контрольной суммы до нуля) |

Общая структура ОЗУ микропроцессорной системы. ОЗУ

|  |  |
| --- | --- |
| 0800 | V\_SUM |
| 0802 | DIAM |
| 0804 | TIME |
| 0808 | T1 |
| 080A | T2 |
| 080C | D\_NUM |
| 080D |  |
| 0FFF | ↑ ОБЛАСТЬ СТЕКА ↑ |
| 1000 | Массив напряжений  с датчика диаметра  (2К) |
| 1FFF | Конец ОЗУ |

**Заключение**

В результате проделанной работы мы получили систему, отбраковывающие бревна, не годные для распиливания и поэтому отбрасывающиеся в сторону. Если бревно нам подходит, то идет подсчет его объема и подсчет всех бревен, прошедших распил. Также считается время наработки.

ПЕРЕЧЕНЬ ЭЛЕМЕНТОВ

|  |  |  |  |
| --- | --- | --- | --- |
| Поз. обозн. | Наименование | Кол. | Примечание |
| DD1 | Микросхема Z-80 | 1 |  |
| DD2, DD4 | Микросхема К155ИР13 | 2 |  |
| DD3, DD5 | Микросхема К537РУ8 | 2 |  |
| DD6 | Микросхема К1118ПВ1 | 1 |  |
| DD7 | Микросхема К546ИР5 | 1 |  |
| DD8 | Микросхема К155ИД3 | 1 |  |
| DD9 | Микросхема К541РЕ1 | 1 |  |
| DD10 | Микросхема К580ВН59А | 1 |  |
| DD11 | Микросхема К155ЛД7 | 1 |  |
| Rd | Резистор МЛТ-5.1К | 1 |  |
| R1, R3 | Резистор МЛТ-10К | 4 |  |
| R 2, Rдоб | Резистор МЛТ-1К | 3 |  |
| R9, R10 | Резистор МЛТ-1-13 кОм А  ТУ ОЖО 467 003 | 2 |  |
| R11, R12 | Резистор МЛТ-1-510 кОм А  ТУ ОЖО 467 003 | 2 |  |
| R8,  R13 – R16 | Резистор МЛТ-0.5-1 кОм А  ТУ ОЖО 467 003 |  |  |
| C1, C8, C9 | Конденсатор МБМ – 16 В – 0,47 мкФ  ТУ УБО 462 014 | 7 |  |
| C6 | Конденсатор МБМ – 16 В – 0,1 мкФ  ТУ УБО 462 014 | 1 |  |
| C7 | Конденсатор МБМ – 16 В – 0,15 мкФ  ТУ УБО 462 014 | 1 |  |
|  |  | 1 |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *2101 420 000 019 ПЭ* | | | |
|  |  |  |  |  |
| *Изм* | *Лист* | *№ докум.* | *Подп.* | *Дата* |
| *Разраб.* | | *Молоков Т* |  |  | Микропроцессорная система управления, предназначенная для использования на лесопильном заводе *Перечень элементов* | *Лит.* | *Лист.* | *Листов* |
|  | | *Данилов Д.* |  |  |  |  |  |
| *Пров.* | | *Мокрецов В.* |  |  | УГТУ – УПИ *РТФ гр. Р–311А*  *Кафедра АУТС* | | |
| *Н.Контр.* | | *Калинин Н.С* |  |  |
| *Утв.* | |  |  |  |