Учреждение Образования
Белорусский Государственный Университет
Информатики и Радиоэлектроники

Факультет вечернего, заочного и дистанционного обучения

Кафедра ЭВМ

Контрольная работа

по дисциплине

«Микропроцессорные средства и системы»

Студента 5 курса 500502 учебной группы

Авсеева Сергея

Минск – 2009 год

1 Техническое задание

Спроектировать систему на базе микроконтроллера MC68HC11. Система должна преобразовывать посредством одного внутреннего АЦП последовательность входных данных. Записывать преобразованные данные по формуле y = x1 * ln(x1) в локальную ОЗУ объемом 17 кбайт и отображать на 4-ех семисегментных индикаторах. Переменная x1 является результатом выборки значений АЦП. Управляющая программа находится в ПЗУ объемом 7 кбайт.

2 Общее описание микроконтроллеров MC68HC11

Семейство 8-разрядных микроконтроллеров M68HC11 является одним из наиболее распространенных в мире семейств микроконтроллеров. Это семейство включает несколько десятков моделей (таблица 1), которые имеют одинаковое процессорное ядро, но отличаются объемом и типом используемой памяти, набором периферийных устройств и рядом других характеристик (тактовая частота, температурный диапазон, тип корпуса).

Таблица 1

Модель	РПЗУ УФ	ПЗУ	РПЗУ	ОЗУ	Вх/вых	АЦП	ШИМ
MC68HC11A8	–	8К	512	256	38	8 каналов, 8 разрядов	–
MC68HC11A1	–	–	512	256	22	8 каналов, 8 разрядов	–
MC68HC11A0	–	–	–	256	22	8 каналов, 8 разрядов	–
MC68HC11E9	–	12К	512	512	38	8 каналов, 8 разрядов	–
MC68HC11E1	–	–	512	512	22	8 каналов, 8 разрядов	–
MC68HC811E2	–	–	2К	256	38	8 каналов, 8 разрядов	–
PC68HC711E9	12К	–	512	512	38	8 каналов, 8 разрядов	–
MC68HC11F1	–	–	512	1К	54	8 каналов, 8 разрядов	–
MC68HC11K4	–	24К	640	768	62	8 каналов, 8 разрядов	4 канала, 8 разрядов
MC68HC711K4	24К	–	640	768	62	8 каналов, 8 разрядов	4 канала, 8 разрядов
MC68HC11G5	–	16К	–	512	66	8 каналов, 10 разрядов	4 канала, 8 разрядов

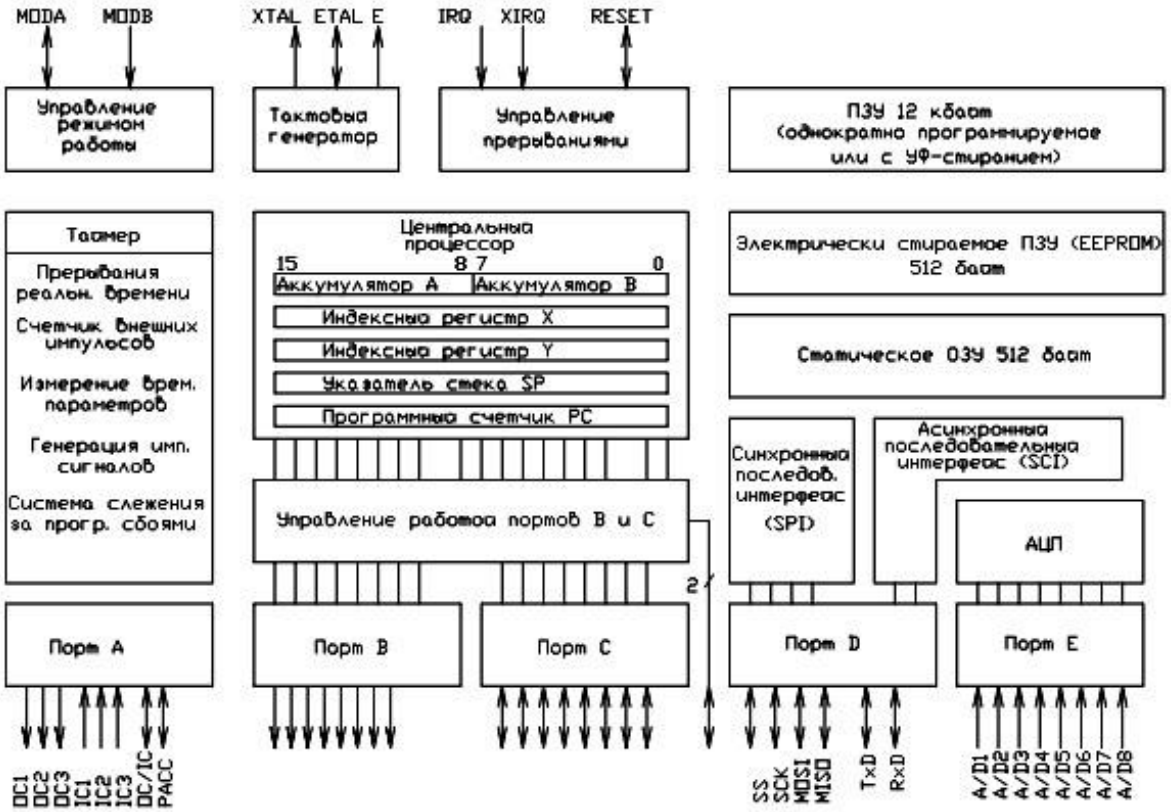

Рисунок 1 - Структурная схема MC68HC11E9

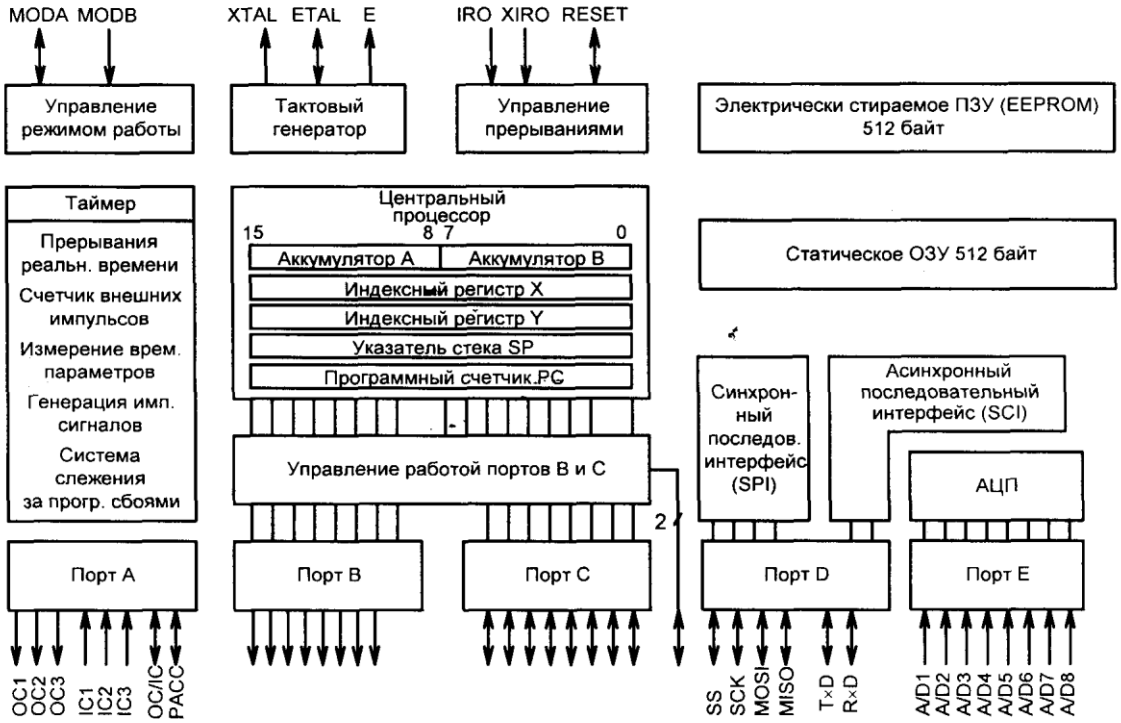

Рисунок 2 - Структурная схема MC68HC11E1

Технология с использованием КМОП-структур с высокой степенью компоновки, использованная для создания MC68HC11, сочетает в себе малые размеры и высокую тактовую частоту по сравнению с КМОП-технологией, одновременно с низким потреблением энергии и высокой устойчивостью по отношению к различным помехам.

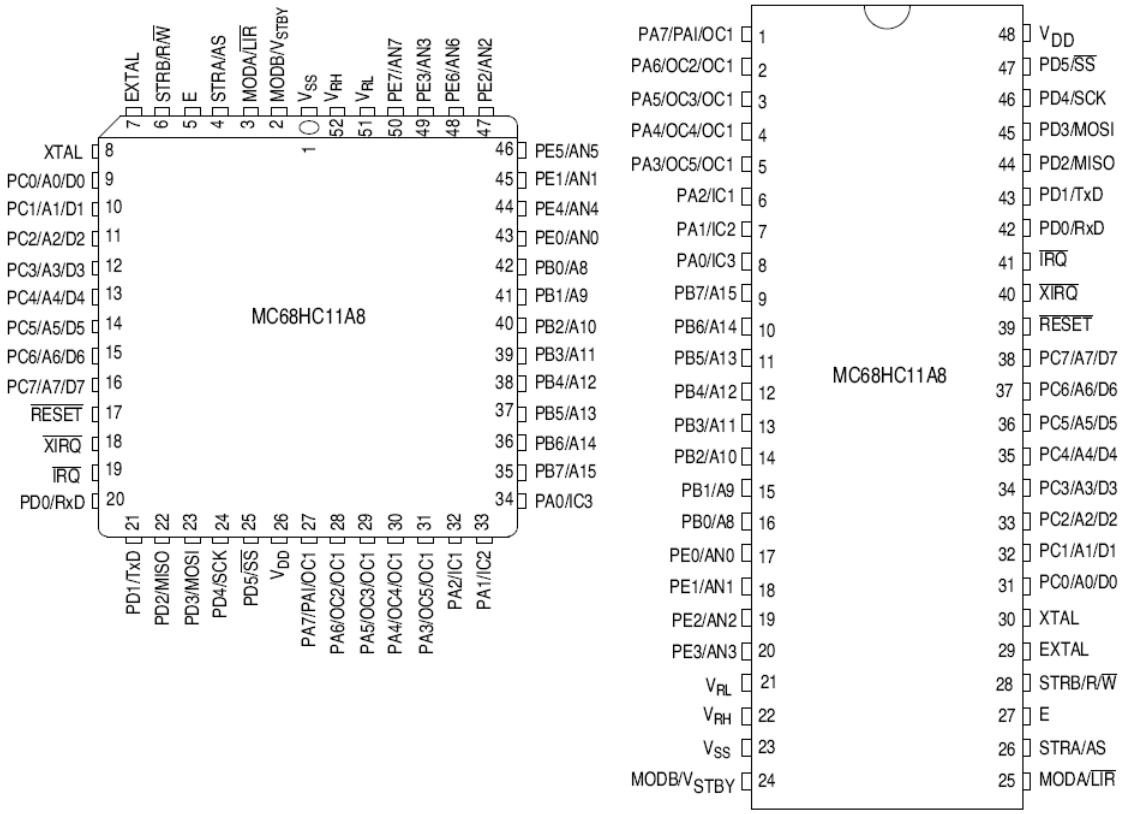

Рисунок 3 – Расположение выводов микросхемы MC68HC11A8

Основные периферийные функции обеспечиваются встроенными схемами микроконтроллера. В их числе: восьмиканальный 8-разрядный АЦП; асинхронный последовательный интерфейс связи; синхронный последовательный периферийный интерфейс; основный 16-разрядный таймер с тремя входными и пятью выходными линиями, поддерживающий прерывания реального времени; 8-разрядный счетчик внешних импульсов для подсчета внешних импульсов или измерения периодов внешних сигналов.

Кроме этого, в состав микроконтроллера входит схема автоматического слежения, предназначенная для защиты системы от ошибок. Эта схема генерирует системный сброс в случае остановки работы или недопустимо низкой частоты тактового генератора.

Схема определения неверного кода операции вызывает немаскируемое прерывание, если в процессе выполнения программы встречается неправильный код команды. Для уменьшения потребления энергии доступно два программно устанавливаемых режима работы - WAIT и STOP.

К типичной области применения относятся контролирующие системы, которые фиксируют аналоговые сигналы, преобразуют их в цифровой код и после обработки отображают на панели ЖКИ или передают главной системе. Наличие таймеров делает идеальным применение данных устройств для промышленного управления: счетчики импульсов, цифровое управление электродвигателями, электротехнические измерения, ручные измерительные инструменты и т.д. Аппаратное умножающее устройство усиливает возможности и предлагает широкий диапазон форматов данных и аппаратную совместимость внутри семейства.

3 Обоснование выбора элементной базы.

В основе системы лежит микроконтроллер DD1 типа MC68HC11E1. Принципиальная схема системы приведена в приложении Б .

В качестве источника тактовых импульсов выбирается внешний кварцевый резонатор ZQ1, конденсаторы C2, C3, подключаемые к выводам EXTAL и XTAL.

В качестве опорного источника для АЦП используется напряжение питания 5В и поступает на выводы VRL, VRH. Накапливающая ёмкость 20пФ встроена в микроконтроллер, поэтому дополнительных ёмкостей устанавливать не требуется. Аналоговый сигнал X1 через разъём XS1 поступает на соответствующую цепь защиты, собранную на стабилитронах VD1,VD2 и фильтрующем конденсаторе C1. На рисунке 4 приведена диаграмма работы внутреннего АЦП. Преобразование входной величины происходит за 12 циклов, запись в регистр ADCTL происходит за 20 циклов.

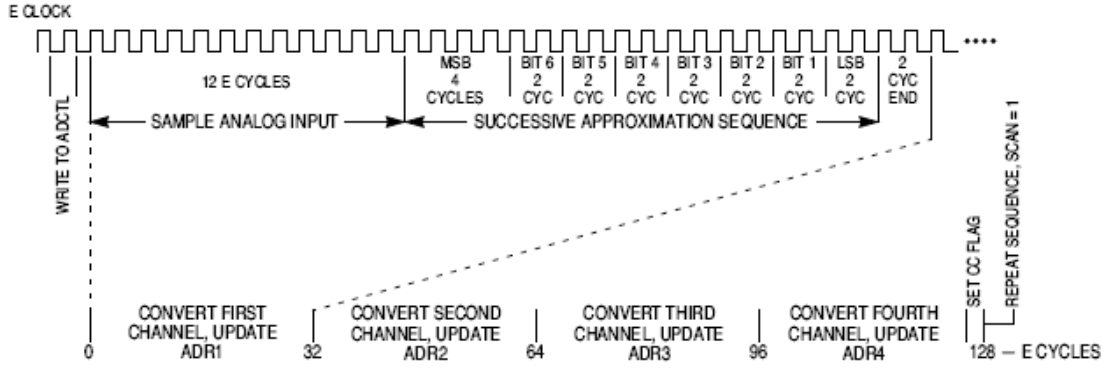

Рисунок 4 – Диаграмма работы микросхемы АЦП

Сегменты светодиодных индикаторов с общим катодом HG1-HG4 подключаются к дешифраторам DD12-DD15, на выходах которых запоминается комбинация сегментов для отображения цифр. Общие катоды индикаторов подключаются к общему проводу GND. Дешифраторы DD12-DD15 также обеспечивают необходимый ток для включения сегментов индикатора. Условное обозначение дешифратора приведено на рисунке 5, а назначение сигналов в таблице 2. Резисторы R15-R42 ограничивают ток через каждый сегмент до 5 мА. Сигналами HGLT0-HGLT3 определяется номер отображаемого разряда числа, таким образом реализовано статическое отображение информации.

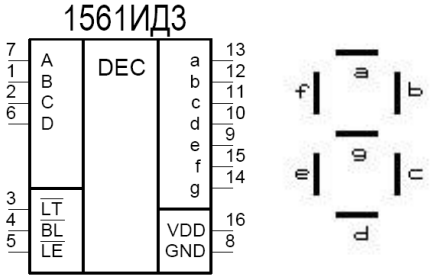

Рисунок 5 –Микросхема дешифратора семисегментного индикатора

Описание сигналов микросхемы 1561ИД3:

a-g - выходы на сегменты;

A-D - информационные входы;

LE - разрешение запоминания (LE=H - пропускание сигнала, LE=L - фиксация сигнала);

LT - вход управления инверсией;

BL - вход блокировки (BL=L - все выходы запираются).

Таблица 2 – Назначение сигналов микросхемы дешифратора 1561ИД3

Код	Входы							Выходы							Символ
	LE	BL	LT	D	C	B	A	a	b	c	d	e	f	g	
-	X	H	L	X	X	X	X	L	L	L	L	L	L	L	
0	H	L	L	L	L	L	L	H	H	H	H	H	H	L	0
1	H	L	L	L	L	L	H	L	H	H	L	L	L	L	1
2	H	L	L	L	L	H	L	H	H	L	H	H	L	H	2
3	H	L	L	L	L	H	H	H	H	H	H	L	L	H	3
4	H	L	L	L	H	L	L	L	H	H	L	L	H	H	4
5	H	L	L	L	H	L	H	H	L	H	H	L	H	H	5
6	H	L	L	L	H	H	L	H	L	H	H	H	H	H	6
7	H	L	L	L	H	H	H	H	H	H	L	L	L	L	7
8	H	L	L	H	L	L	L	H	H	H	H	H	H	H	8
9	H	L	L	H	L	L	H	H	H	H	H	L	H	H	9
10	H	L	L	H	L	H	L	L	L	L	L	L	L	L	Пусто
11	H	L	L	H	L	H	H	L	L	L	L	L	L	L	Пусто
12	H	L	L	H	H	L	L	L	L	L	L	L	L	L	Пусто
13	H	L	L	H	H	L	H	L	L	L	L	L	L	L	Пусто
14	H	L	L	H	H	H	L	L	L	L	L	L	L	L	Пусто
15	H	L	L	H	H	H	H	L	L	L	L	L	L	L	Пусто
-	L	L	L	X	X	X	X	Сохраняется предыдущее состояние на выходах							
-	X	X	H	X	X	X	X	На выходах инверсная комбинация							

Таблица 3 – Электрические характеристики микросхемы дешифратора 1561ИД3

Наименование параметра	Значение
Параметры (T=+25) при питании	E=+9
Выходной ток логического 0, мА	2.0
При выходном напряжении, В	-
Выходной ток логической 1, мА	2.0
При выходном напряжении, В	7.0
Задержки распространения, нс	850
Напряжение питания, В	3-15

В блоке локальной ПЗУ применены микросхемы объемом 1, 2, 4 кбайт (DD2, DD5, DD8), управление которыми осуществляется дешифраторами DD9, DD10. В блоке локальной ОЗУ применены микросхемы памяти объемом 1 и 16 кбайт (DD4, DD7). В блоке памяти обеспечивается переключение микросхем памяти в соответствии с адресацией по рисунку 6

Рисунок 6- Карта памяти

Резисторы R5-R9 служат для подтягивания сигналов управления микросхемами к неактивному состоянию в случае выбора работы с микросхемами памяти.

Структурная схема микросхемы ОЗУ приведено на рисунке 7.

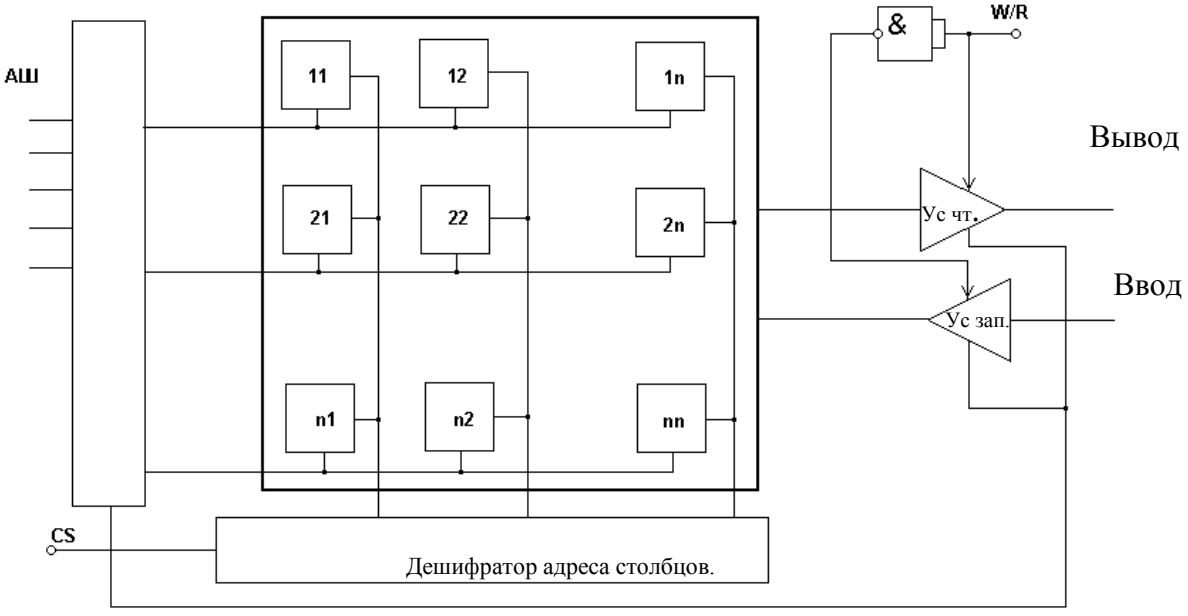

Рисунок 7- Организация ИМС ОЗУ

Оперативные ЗУ предназначены для хранения переменной информации и имеют практически одинаковое быстродействие при считывании и записи. По способу хранения информации ОЗУ делятся на два основных типа: статические и динамические. Статические запоминающие элементы могут хранить информацию сколь угодно долго, пока подается электропитание. Динамические запоминающие элементы, напротив, способны хранить информацию только непродолжительное время. Поэтому для хранения

информации её нужно периодически обновлять, или, другими словами регенерировать. Для обоих типов оперативных ЗУ существует множество различных схем. Их разнообразие отражает не только множество технологий (ТТЛ, *n*-МОП, КМОП, ЭСЛ и т.д.) и конструкций, но ещё и разнообразие требований, предъявляемым к модулям памяти в отношении быстродействия, емкости, плотности упаковки элементов и потребляемой мощности.

Статические ЗУ с произвольной выборкой (Random Access Memory) строятся на триггерах с непосредственными связями, которые могут неограниченно долго хранить информацию при включенном питании. Эти ОЗУ очень просты в эксплуатации, обладают высокой помехоустойчивостью, не требуют дорогих и сложных схем обслуживания, благодаря чему достигается умеренная стоимость всей системы памяти. При интегральной реализации статических ОЗУ используются два вида запоминающих матриц: накопители повышенного быстродействия (время цикла менее 100 нс) без схем дешифрации со средней степенью интеграции в БИС (до 256 бит); накопители среднего быстродействия (время цикла 300-1000 нс) с повышенной информационной ёмкостью от 256 до 16384 бит со схемами дешифрации. Статические ОЗУ в зависимости от принципа построения накопителя имеют словарную или матричную организацию. При словарной организации ОЗУ обращение производится одновременно к запоминающим элементам нескольких разрядов, соответствующих некоторой части слова или всего слова. Основными достоинствами ОЗУ со словарной организацией является простота базовой ячейки, и минимальное число шин управления, необходимых для реализации накопителя. Важное значение имеет также и то обстоятельство, что при словарной организации матрицы БИС в виде m одноразрядных слов удается обеспечить минимальную мощность рассеяния в режиме записи и считывания.

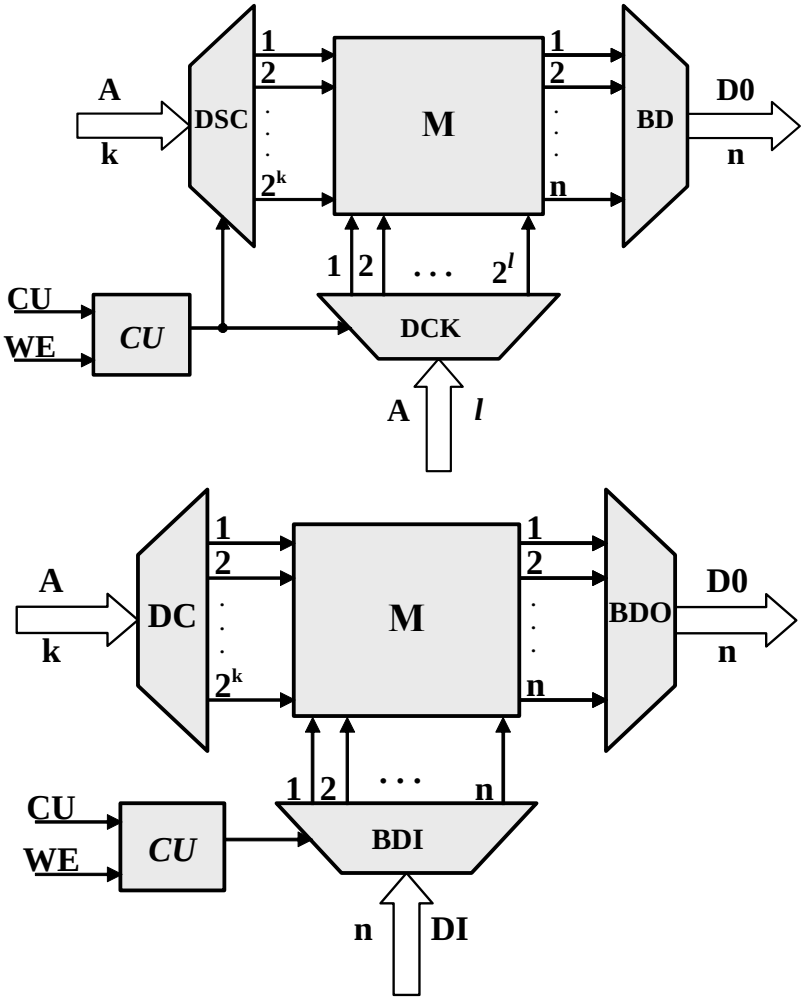

Рисунок 8 - Словарная организация БИС ОЗУ

Обобщенная структура БИС со словарной организацией матрицы приведена на рисунке 8. Код адреса n-разрядного слова подается на адресный дешифратор, который выбирает нужное слово. Адресный усилитель возбуждает соответствующую словарную шину и слово, код которого поступает на входные разрядные шины, записывается в выбранную строку матрицы согласно коду адреса. Аналогично, с помощью разрядных усилителей производится считывание выбранного слова в выходной регистр.

При матричной организации БИС возможно обращение к любому ЗЭ накопителя независимо от других элементов, расположенных на той же БИС. Микросхемы с матричной организацией называют также ОЗУ с разрядной организацией или с двукоординатной выборкой.

При разработке ОЗУ большой ёмкости (≥16 Кбит) применяется микросхемы ОЗУ динамического типа, в которых увеличение ёмкости достигается за счет уменьшения числа элементов и как следствие уменьшение занимаемой площади. Уменьшение числа элементов происходит при использовании динамических запоминающих ячеек, в которых информация хранится в виде заряда соответствующих ёмкостей. Ток утечки обратно смещенного p-n перехода имеет значение не более 10^{-10} А, а ёмкость накопительного конденсатора не превышает 0,1-0,2 пФ, следовательно постоянная времени разряда конденсатора t≥1 мс. Поэтому для выдачи состояния низкого или высокого уровня сигнала на выходе БИС необходимо осуществлять периодическое восстановление информации (или её регенерацию) с периодом t_{REF} ≤1÷2 мс.

Таким образом, главные отличия динамических устройств памяти от статических заключаются в следующем: отсутствует источник питания запоминающих ячеек; необходимы логические схемы, обеспечивающие регенерацию ячеек; обрамление требует более сложных схем; максимальная простота схемы накопителя, для обеспечения минимально занимаемой площади; меньшая потребляемая мощность.

Итак, проведя сравнительный анализ принципов работы и основных характеристик статических и динамических устройств памяти выберем ОЗУ статического типа со словарной организацией.

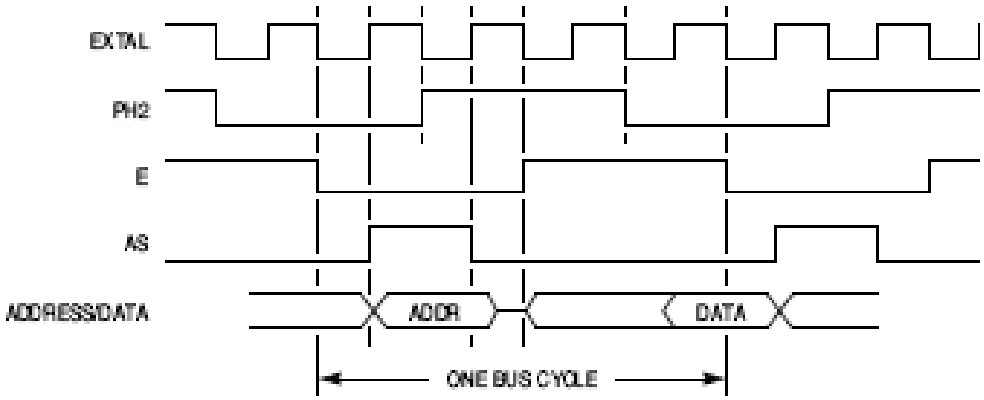

Рисунок 9 - Диаграмма чтения и записи в память

В настоящее время разработаны и выпускаются ПЗУ нескольких типов:

- ПЗУ масочного типа;
- программируемые ПЗУ;
- электрически программируемые ПЗУ;
- электрически программируемые ПЗУ с ультрафиолетовым стиранием.

Масочные ПЗУ – микросхемы, в которых информация записывается при изготовлении с фиксированным рисунком межсоединений, определяемым маской (шаблоном). В ПЗУ запоминающие элементы объединены в двухкоординатную матрицу, образованную при пересечении совокупности входных (чисел) и выходных (разрядов) информационных шин. В местах пересечений шин могут быть включены диоды,

биполярные транзисторы и МОП-транзисторы. Наибольшее распространение получили ПЗУ на МОП-транзисторах ввиду технологической простоты и связанной с этим возможностью получения высокой степени интеграции, а так же малой потребляемой мощностью. Запись информации в масочное ПЗУ производится с помощью сменного заказного фотошаблона. Документом, определяющим хранимую в накопителе информацию, является карта заказа на данную микросхему. Изготовление маски довольно дорого, но с помощью одной маски можно запрограммировать любое число модулей памяти. Следовательно, масочные ПЗУ рентабельны при крупносерийном производстве.

Постоянные запоминающие устройства, допускающие однократное программирование у заказчика – это микросхемы, в которых состояние ячеек можно задать уже после изготовления устройства (создав либо разрушив перемычки). Наибольшее распространение получили перемычки в виде плавких вставок (например, из нихрома или поликремния), которые можно избирательно пережечь, с помощью внешнего источника тока. Накопитель ППЗУ представляет собой матрицу на биполярных транзисторах с плавкими перемычками, включенными последовательно с эмиттерами транзисторов, т.е. функциональная схема БИС ППЗУ аналогична схеме масочного ПЗУ.

Программирование БИС ППЗУ разных серий производится на специальных устройствах-программаторах.

Постоянные запоминающие устройства, допускающие многократное программирование и сохраняющие информацию при отключении питания (Errasable-Programmable-Read-Only-Memory – стираемая программируемая память только со считыванием) – микросхемы, использующие элементы коммутации, которые можно устанавливать в одно (замкнутое) состояние избирательно, а в другое (разомкнутое) – коллективно. Программирование таких ПЗУ сводится сначала к коллективной установке всех перемычек в одно состояние, что равносильно стиранию ранее записанной информации и последующей поочередной установки нужных перемычек в другое состояние.

Электрически программируемые ПЗУ характеризуются сочетанием положительных качеств ПЗУ – энергонезависимым хранением информации и высокой удельной плотностью её записи с возможной многократной сменой информации, как в ОЗУ.

Микросхемы со стиранием ультрафиолетом представляют собой РПЗУ на основе лавинно-инжекционных МОП-транзисторов с плавающим затвором, в которых запись информации осуществляется электрическим способом, а для стирания информации требуется облучение ультрафиолетовым излучением.

Мы для своего микропроцессорного комплекта выберем наиболее простой тип ПЗУ это программируемое постоянное запоминающее устройство серии 27xx с тремя состояниями на выходе.

Питание системы осуществляется от стабилизированного источника питания +5В, построенном на микросхеме линейного стабилизатора напряжения DA1. Кнопка S1 служит для ручного сброса микроконтроллера в исходное состояние.

В приложении А приведён листинг управляющей программы.

Система производит вывод результатов вычислений в виде чисел с плавающей точкой, поэтому для увеличения точности вывода можно увеличить разрядность индикации до 8-ми.

Приложение А

Листинг управляющей программы

```
HPRIO equ 0x103C
PIOC equ 0x1002
PORTA equ 0x1000
PORTC equ 0x1005
PORTD equ 0x1008
PORTE equ 0x100A
ADCTL equ 0x1030
ADR1 equ 0x1031

      .area data
_digit_out::            ; Содержимое разрядов индикаторов
      .blkb 4
      .area idata
      .byte 0,0,0,0
      .area data
_adc_val::        ; Значение АЦП
      .blkb 2
      .area idata
      .word 0
      .area bss
_buf1::
      .blkb 2           ; Адрес __digit_out
_rev::
      .blkb 26    ; Временный массив результатов преобразования в ASCII
_rem::
      .blkb 2           ; Остаток десятичного деления
_s::
      .blkb 2           ; Указатель на выходной разряд
_chan::
      .blkb 1           ; Номер считываемого канала АЦП
_X1::
      .blkb 1           ; Переменная x1
_X2::
      .blkb 1           ; Переменная x2
_X3::
      .blkb 1           ; Переменная x3
_value::
      .blkb 4           ; Временное значение результатов вычислений
_address::
      .blkb 2           ; Смещение буфера ОЗУ для хранения результатов
вычислений
_i::
      .blkb 1           ; Переменная для циклов
_buf::
      .blkb 24000 ; Буфер ОЗУ для хранения результатов вычислений
__fpacc1_exp::
      .blkb    1  ; Аккумулятор для экспоненты 1
__fpacc1_mantissa::
      .blkb    3  ; Аккумулятор для мантиссы 1
__fpacc1_mantissa_sign::
      .blkb    1  ; Знак мантиссы 1
__fpacc2_exp::
      .blkb    1  ; Аккумулятор для экспоненты 2
__fpacc2_mantissa::
      .blkb    3  ; Аккумулятор для мантиссы 2
__fpacc2_mantissa_sign::
      .blkb    1  ; Знак мантиссы 2
_d::
      .blkw 1
_y::
      .blkw 1
```

```
_x::
      .blkw 1
; Таблица полиномов
LNTBL::
         .byte    0x3E,0x1D,0x89,0xD9     ;  2/13
         .byte    0x3E,0x3A,0x2E,0x8C     ;  2/11
         .byte    0x3E,0x63,0x8E,0x39       ;  2/9
         .byte    0x3E,0x92,0x49,0x25       ;  2/7
         .byte    0x3E,0xCC,0xCC,0xCD   ;  2/5
         .byte    0x3F,0x2A,0xAA,0xAB   ;  2/3
         .byte    0x40,0x00,0x00,0x00       ;  2/1
         .byte    0xFF                    ; Признак конца таблицы
; Константы
MONE::     .byte    0xBF,0x80,0x00,0x00  ;    -1.0
NLN2::     .byte    0x3F,0x31,0x72,0x18    ;   LN(2)
ONE::    .byte    0x3F,0x80,0x00,0x00

      .area text
L30:
      .word 0,10
L24:
      .byte 48,0
L23:
      .word 0,0

      .area text
      .org # 0xF000
_start::
      jsr __enterb
      .byte 0x8c
      ldab #96
      stab HPRIO
      ldab #16
      stab PIOC
      jmp L5
L4:               ; Начало цикла записи результатов в ОЗУ
      ldd #0
      std _address
      jmp L10
L7:
      jsr _read_adc ; Чтение АЦП
      stab _X1
      ldab _X1
      clra
      jsr __d2lreg
      jsr __long2fp
      ldd 0,x
      addd #10
      xgdy
      jsr __fromfp1
      ldd 0,x
      addd #10
      xgdy
      jsr __tofp1
      pshy
      pshy
      tsy
      jsr __fromfp1
      jsr _log    ; Вычисление логарифма
      puly
      puly
      ldd 0,x
      addd #6
      xgdy
      jsr __fromfp1
```

```
      ldd 0,x
      addd #10
      xgdy
      jsr __tofp1
      ldd 0,x
      addd #6
      xgdy
      jsr __tofp2
      jsr __fpmul ; Умножение на X1
      jsr __fp2long
      ldd 0,x
      addd #2
      xgdy
      jsr __lreg2y
      ldd 0,x
      addd #2
      xgdy
      jsr __ly2reg
      ldd _address
      lsld
      lsld
      addd #_buf
      xgdy
      jsr __lreg2y
      ldd 0,x
      addd #2
      xgdy
      jsr __ly2reg
      ldy #_value
      jsr __lreg2y
      jsr _disp_out ; Вывод результата на индикаторы
L8:
      ldd _address
      addd #1
      std _address
L10:
      ldd _address
      cpd #24000
      bhs X1
      jmp L7
X1:
L5:
      jmp L4
X0:
L3:
      xgdx
      addd #12
      xgdx
      txs
      pulx
      rts

; Подпрограмма чтения данных из АЦП
_read_adc::
      ldab _chan  ; Номер обрабатываемого канала
      stab ADCTL  ; Запуск преобразования
L12:
      ldy # ADCTL
      brclr 0,y,#128,L12 ; Ожидание завершения преобразования
      ldab ADR1
      clra
      rts

; Подпрограмма вывода на индикаторы
_disp_out::
```

```
      jsr _ltoa   ; Преобразование результата вычисления в ASCII
      clr _i            ; Начало цикла перебора разрядов
      bra L19
L16:
      ldab _i
      clra
      addd #_digit_out
      xgdy
      ldab 0,y
      subb #48    ; Приведение к двоичному коду
      stab PORTD
      ldab _i
      lslb
      lslb
      lslb
      addb #64
      stab PORTA
L17:
      ldab _i
      addb #1
      stab _i
L19:
      ldab _i
      cmpb #6
      blo L16
      clr PORTA
L15:
      rts

_log::
      pshx
      tsx
      ldab #4
      abx
      jsr   __to_fpacc1
      bsr   FLTLN
      pulx
      rts

FLTLN::
         ldaa    __fpacc1_mantissa_sign    ;Проверка на знак
         beq     LN1     ;Переход если положительное число
LN0:      ldaa   #LNNEGERR    ; Отрицательное или равно 0
         sec      ;Признак ошибки
         rts      ;Выход из подпрограммы
LN1:      ldaa    __fpacc1_exp ; Проверка на 0
         beq     LN0
         jsr     __push_fpacc2 ;Сохранить аккумулятор в стеке
         pshx     ;Создание стекового фрейма для хранения промежуточных
вычислений
         pshx
         tsx      ;Создать указатель для промежуточных значений
         jsr     __from_fpacc1 ;Сохранить аргумент
         clra
         ldab    __fpacc1_exp
         subb    #0x7F         ; Преобразование экспоненты
         bpl     LN2           ;Переход, если экспонента положительная
         coma
LN2:      jsr    int_to_fp    ; Преобразование целого числа в число с
плавающей точкой
         ldx     #NLN2   ;указатель на константу LN(2)
         jsr     __to_fpacc2   ;Записать в аккумулятор 2
         jsr     __fpmul
         tsx
         jsr     __to_fpacc2   ;Загрузить аргумент
```

```
        jsr     __from_fpacc1 ;сохранить частичный результат
        ldaa    #0x7F
        staa    __fpacc2_exp  ; Мантисса с экспонентой = 0
        pshx                  ; Резервирование места в стеке для мантиссы
        pshx
        tsx
        jsr     __from_fpacc2 ; Сохранить мантиссу (M)
        ldx     #ONE          ; Указатель на константу 1
        jsr     __to_fpacc1   ; Сохранить в аккумуляторе 1
        jsr     __fpadd       ;(M+1)
        tsx                   ;Вернуть указатель на мантиссу
        jsr     __to_fpacc2   ;(M)
        jsr     __from_fpacc1 ;(M+1)
        ldx     #MONE         ; Указатель на константу -1
        jsr     __to_fpacc1   ; Сохранить в аккумуляторе 1
        jsr     __fpadd       ;(M-1)
        tsx                   ; (M+1)
        jsr     __to_fpacc2   ; Сохранить в аккумуляторе 2
        jsr     __fpdiv       ;(M-1)/(M+1)
        tsx                   ; Указатель на временный результат
        jsr     __from_fpacc1 ; Считать содержимое аккумулятора 1
        jsr     __fp_xfer_acc1_to_acc2
        jsr     __fpmul             ;Квадрат значения
        ldx     #LNTBL        ;Указатель на полиномиальный коэффициенты
        jsr     POLYNOM ;Вычислить полином
        tsx                   ;Получить указатель на результат (M-1)/(M+1)
        jsr     __to_fpacc2   ; Записать результат в аккумулятор 2
        jsr     __fpmul       ; умножение мантиссы
        pulx
        pulx
        tsx
        jsr     __to_fpacc2
        jsr     __fpadd        ;Сложить экспоненты
        pulx
        pulx
        jsr     __pull_fpacc2
        clrb
        rts

POLYNOM::
        xgdx                  ; Сохранить указатель на таблицу в D
        ldx     __fpacc1_exp  ; Сохранить аргумент в стеке
        pshx
        ldx     __fpacc1_mantissa+1
        pshx
        xgdx
        ldaa    __fpacc1_mantissa_sign
        psha
        pshx
        ldd     #0                  ; Инициализировать аккумулятор
        std     __fpacc1_exp
        std     __fpacc1_mantissa+1
        std     __fpacc1_mantissa+2
        bra     POLY1
POLY2:   tsx
        ldaa    2,x
        staa    __fpacc2_mantissa_sign
        ldd     3,x
        std     __fpacc2_mantissa+1
        ldd     5,x
        std     __fpacc2_exp
        jsr     __fpmul
POLY1:   tsx
        ldx     0,x
        jsr     __to_fpacc2
```

```
        jsr     __fpadd
        pulx
        ldab    #4                  ; Увеличить указатель на таблицу
        abx
        pshx
        ldaa    0,x             ; Проверка окончания таблицы
        coma                    ; Если FF, то завершение
        bne     POLY2
        tsx
        ldab    #7
        abx
        txs
        rts

__push_fpacc2::
        pulx
        pshx
        pshx
        xgdx
        tsx
        pshb
        psha
        jmp     __from_fpacc2

__from_fpacc2::
        ldd     __fpacc2_mantissa+1
        std     2,x
        ldd     __fpacc2_exp
        lslb
        cmpb    __fpacc2_mantissa_sign
        rora
        rorb
        std     0,x
        rts

__to_fpacc1::
        clr     __fpacc1_mantissa_sign    ; Положительное число в аккумуляторе
        ldd     2,x                       ; Загрузить младшие 16 бит мантиссы
        std     __fpacc1_mantissa+1 ; Записать в аккумулятор
        ldd     0,x                       ; Получить экспоненту и старший байт
мантиссы
        lsld                        ; Сдвиг с переносом
        bcc     GETFP11       ; Переход, если число положительное
        com     __fpacc1_mantissa_sign    ; Установить знак
GETFP11:  staa   __fpacc1_exp ;  Сохранить экспоненту и проверить на 0
        beq     GETFP12 ; Переход если 0
        sec                     ; Установить перенос для сдвига
        rorb                    ; Нормализация мантиссы
GETFP12:  stab   __fpacc1_mantissa  ; Сохранить в аккумуляторе
        clrb                        ; сброс ошибок
        rts

__to_fpacc2::
        clr     __fpacc2_mantissa_sign    ; Положительное число в аккумуляторе
        ldd     2,x                 ; Загрузить младшие 16 бит мантиссы
        std     __fpacc2_mantissa+1 ; Записать в аккумулятор
        ldd     0,x                 ; Получить экспоненту и старший байт
мантиссы
        lsld                    ; Сдвиг с переносом
        bcc     GETFP21       ; Переход, если число положительное
        com     __fpacc2_mantissa_sign    ; Установить знак
GETFP21:  staa   __fpacc2_exp ;  Сохранить экспоненту и проверить на 0
        beq     GETFP22       ; Переход если 0
        sec                     ; Установить перенос для сдвига
        rorb                    ; Нормализация мантиссы
```

```
GETFP22:  stab   __fpacc2_mantissa ;  Сохранить в аккумуляторе
          rts

; Подпрограмма преобразования результатов вычислений в символьный вид
_ltoa::
      jsr __enterb
      .byte 0x84
      ldd #_digit_out
      std _buf1
      ldy #_value
      jsr __ly2reg
      ldy #L23
      jsr __ly2reg2
      jsr __lcmp  ; Проверка выводимого значения на 0
      bne L21
      ldd #L24
      std _s
      bra L22
L21:
      clr _rev+25
      ldd #_rev+25
      std _s
      bra L28
L27:
      ldy #_value
      jsr __ly2reg
      ldy #L30
      jsr __ly2reg2
      jsr __lmod
      jsr __lreg2d
      std _rem
      ldd _rem
      cpd #10
      bge L31
      ldd _s
      addd #-1
      std 2,x
      std _s
      ldd _rem
      addd #48
      ldy 2,x
      stab 0,y
L31:
      ldy #_value
      jsr __ly2reg
      ldy #L30
      jsr __ly2reg2
      jsr __ldiv
      ldy #_value
      jsr __lreg2y
L28:
      ldy #_value
      jsr __ly2reg
      ldy #L23
      jsr __ly2reg2
      jsr __lcmp
      bne L27
L22:
      ldd #6
      pshb
      psha
      ldd _s
      pshb
      psha
      ldd _buf1
```

```
      jsr _memcpy
      puly
      puly
L20:
      inx
      inx
      inx
      inx
      txs
      pulx
      rts

; Подпрограмма умножения чисел с плавающей точкой
__fpmul::
      pshx
      bsr fpmul
      pulx
      rts
fpmul::
         tst    __fpacc1_exp     ;Проверка аккумулятора 1 на 0
         beq    FPMULT3
         tst    __fpacc2_exp     ; Проверка аккумулятора 2 на 0
         bne    FPMULT8
FPMULT3: ldd    #0
         staa   __fpacc1_mantissa_sign
         std    __fpacc1_exp
         std    __fpacc1_mantissa+1
         rts
FPMULT8: ldaa   __fpacc1_mantissa_sign
         eora   __fpacc2_mantissa_sign
         staa   __fpacc1_mantissa_sign
         ldaa   __fpacc1_exp
         adda   __fpacc2_exp
         bpl    FPMULT1
         bcc    FPMULT2
__fp_return_max::  ldaa   #OVFERR
__fp_return_div0:: ldx    #0xFFFF
         bra    FPMULT7
FPMULT1: bcs    FPMULT2

__fp_return_zero::
         ldaa   #UNFERR
         ldx    #0
         stx    __fpacc1_mantissa+2
FPMULT7: stx    __fpacc1_exp
         stx    __fpacc1_mantissa+1
         sec                  ;FLAG ERROR.
         rts                  ;RETURN.
FPMULT2: adda   #0x82
         staa   __fpacc1_exp  ; Сохранить новую экспоненту
         ldx    #0
         pshx
         pshx
         tsx
         jsr    UMULT         ;Умножить мантиссы
         tst    0,x                 ;Проверка необходимости нормализации
результата
         bmi    FPMROUND
         rol    3,x
         rol    2,x
         rol    1,x
         rol    0,x
         dec    __fpacc1_exp  ;Уменьшение экспоненты
FPMROUND:tst    3,x
         bpl    FPMULT4
```

```
          ldaa   2,x
          inca
          staa   2,x
          bne    FPMULT4
          ldab   #1
          addd   0,x
          bcc    FPMULT5
          rora
          inc    __fpacc1_exp
FPMULT5:  std    0,x
FPMULT4:  pulx
          stx    __fpacc1_mantissa
          pula
          staa   __fpacc1_mantissa+2
          ins
          tst    __fpacc1_exp
          beq    __fp_return_zero
          clrb
          rts

UMULT:
          ldaa   __fpacc2_mantissa+2
          ldab   __fpacc1_mantissa+2
          mul
          staa   1,x          ;Сохранение промежуточного результата умножения
          ldaa   __fpacc2_mantissa+1
          ldab   __fpacc1_mantissa+2
          mul
          addd   0,x
          std    0,x
          ldaa   __fpacc2_mantissa+2
          ldab   __fpacc1_mantissa+1
          mul
          addd   0,x
          staa   3,x
          bcc    UMULT1
          inc    2,x
UMULT1:   clr    0,x
          clr    1,x
          ldaa   __fpacc2_mantissa
          ldab   __fpacc1_mantissa+2
          mul
          addd   2,x
          std    2,x
__fp_remcomp::
              ldaa   __fpacc2_mantissa+1
          ldab   __fpacc1_mantissa+1
          mul
          addd   2,x
          std    2,x
          bcc    UMULT2
          inc    1,x
UMULT2:   ldaa   __fpacc2_mantissa+2
          ldab   __fpacc1_mantissa
          mul
          addd   2,x
          std    2,x
          bcc    UMULT3
          inc    1,x
UMULT3:   ldaa   __fpacc2_mantissa
          ldab   __fpacc1_mantissa+1
          mul
          addd   1,x
          std    1,x
          bcc    UMULT4
```

```
        inc    0,x
UMULT4: ldaa   __fpacc2_mantissa+1
        ldab   __fpacc1_mantissa
        mul
        addd   1,x
        std    1,x
        bcc    UMULT5
        inc    0,x
UMULT5: ldaa   __fpacc2_mantissa
        ldab   __fpacc1_mantissa
        mul
        addd   0,x
        std    0,x
        rts

; Подпрограмма резервирования стека для переменных
__enterb::
      puly        ; В регистре y находится адрес возврата
      brclr 0,y,#0x40,no_pshd
      pshb
      psha
no_pshd:
      pshx        ; Сохраняем указатель предыдущего фрейма данных
      tsx         ; адрес стека в X
      ldab 0,y    ; Получить размер фрейма
      clra
      andb #0x3F  ; Сбрасываем 2 младших бита
      pshb
      psha        ; Записываем размер фрейма в стек
      xgdx        ; адрес стека в D
      tsx
      subd 0,x    ; вычесть размер фрейма
      xgdx
      txs         ; Записать в SP новый адрес
      stx 0,x           ; сохранить IX
      brclr 0,y,#0x80,no_lregb
      pshx
      pshx
      pshx
      pshx
no_lregb:
      jmp   1,y

__d2lreg::
      pshb
      psha
      xgdx
      subd #8
      xgdx
      clr 4,x
      clr 5,x
      pula
      pulb
      std 6,x
tneg:
      bge done1
      com 4,x
      com 5,x
done1:
      ldab #8
      abx
      rts

__long2fp::
      jsr __pshfp2      ; сохранение fp2 в стеке
```

```
      pshx
      pshy
      pshx        ; выделить пространство в стеке
      pshx
      tsy
      jsr __lreg2y      ; создать аргумент
      jsr __c_long2fp
      pulx
      pulx
      puly
      pulx
      jsr   __pulfp2; вывод fp2 из стека
      rts

__pshfp2::
      sty _y
      ldy #__fpacc2_exp
      bra psh
psh:
      stx _x
      std   _d
      tsx
      ldx 0,x           ; rts
      pula        ; pop rts
      pulb
      ldd 3,y
      pshb
      psha
      ldd 1,y
      pshb
      psha
      ldab 0,y
      pshb
      ldy _y
      ldd _d
      pshx
      ldx _x
      rts

__pulfp2::
      sty _y
      ldy #__fpacc2_exp
      bra pul
pul:
      stx _x
      std _d
      tsx
      ldab 2,x
      stab 0,y
      ldd 3,x
      std 1,y
      ldd 5,x
      std 3,y
      ldx 0,x           ; get rts
      puly  ; pop rts
      pulb  ; pop fp1
      puly
      puly
      ldy _y
      ldd _d
      pshx  ; rts
      ldx _x
      rts
__fromfp1::
      pshx
```

```
      pshy
      pulx
      jsr __from_fpacc1
      pulx
      rts
__from_fpacc1::
         ldd    __fpacc1_mantissa+1
         std    2,x
         ldd    __fpacc1_exp
         lslb
         cmpb   __fpacc1_mantissa_sign
         rora
         rorb
         std    0,x
         rts

__fp2long::
      pshx
      pshx        ; выделить стек
      pshx
      tsy
      jsr __fromfp1
      jsr __c_fp2long
      pulx
      pulx
      pulx
      rts
__lreg2y::
      pshb
      psha
      xgdx
      subd #8
      xgdx
      ldd 4,x
      std 0,y
      ldd 6,x
      std 2,y
      ldab #8
      abx
      pula
      pulb
      rts
__ly2reg::
      pshb
      psha
      xgdx
      subd #8
       xgdx
      ldd 0,y
      std 4,x
      ldd 2,y
      std 6,x
      ldab #8
      abx
      pula
      pulb
      rts

; Подпрограмма копирования чисел с плавающей точкой
__fpmov::
      pshb
      psha
      pshx
      pshy
      ldx   #__fpacc1_exp
```

```
      ldy   #__fpacc2_exp
      ldaa 0,x
      staa 0,y
      ldd 1,x
      std 1,y
      ldd 3,x
      std 3,y
      puly
      pulx
      pula
      pulb
      rts

__tofp1::
      pshx
      pshy
      pulx
      jsr __to_fpacc1
      pulx
      rts

; Подпрограмма деления чисел с плавающей точкой
__fpdiv::
      pshx
      bsr fpdiv
      pulx
      rts

fpdiv:
         ldaa   __fpacc2_mantissa_sign
         eora   __fpacc1_mantissa_sign
         staa   __fpacc1_mantissa_sign
         tst    __fpacc2_exp
         bne    FLTDIV1
         ldaa   #DIV0ERR
         jmp    __fp_return_div0
FLTDIV1:  tst    __fpacc1_exp
         bne    FLTDIV2
         clrb
         clr    __fpacc1_mantissa_sign
         rts
FLTDIV2:  ldaa   __fpacc1_exp
         suba   __fpacc2_exp
         bhs    FLTDIV8
         bmi    FLTDIV6
         jmp    __fp_return_zero
FLTDIV8:  bmi    FLTDIV7
FLTDIV6:  adda   #0x7E
         staa   __fpacc1_exp
         ldd    __fpacc1_mantissa
         subd   __fpacc2_mantissa
         bmi    FLTDIV3
         bne    FLTDIV4
         ldaa   __fpacc1_mantissa+2
         suba   __fpacc2_mantissa+2
         blo    FLTDIV3
FLTDIV4:  ldd    __fpacc1_mantissa
         lsrd
         ror    __fpacc1_mantissa+2
         std    __fpacc1_mantissa
         inc    __fpacc1_exp
         bne    FLTDIV3
FLTDIV7:  jmp    __fp_return_max
FLTDIV3:  ldd    __fpacc1_mantissa
         ldx    __fpacc2_mantissa
```

```
         inx
         bne    FLTDIV5
         dex
FLTDIV5:  fdiv
         pshy
         ldy    __fpacc1_mantissa
         stx    __fpacc1_mantissa
         pshx
         ldx    #0
         pshx
         pshx
         tsx
         ldaa   __fpacc2_mantissa+2
         ldab   __fpacc1_mantissa+1
         mul
         staa   3,x
         jsr    __fp_remcomp
         sty    __fpacc1_mantissa
         ldd    __fpacc1_mantissa+1
         subd   1,x
         ldx    __fpacc2_mantissa
         fdiv
         stx    __fpacc1_mantissa+1
         clr    __fpacc1_mantissa
         pulx
         pulx
         pula
         pulb
         puly
         addd   __fpacc1_mantissa
         std    __fpacc1_mantissa
         clrb
         rts

      .org 0xffd6
_interrupt_vectors::    ; Таблица векторов прерываний
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word 65535
      .word _start
```